APLICACIÓN E-LEARNING

Plataforma de aprendizaje de lenguas flexivas

Alberto Daimiel Blanco

Mounir Hichou Al Luch

Darío Simón de Vega

Luis Zorrilla Suárez

Curso 2016-2017

Profesores:

Antonio Sarasa Cabezuelo

Ana María Fernández-Pampillón Cesteros

Trabajo de Fin de Grado 2016-2017

Plataforma de aprendizaje de lenguas flexivas

Universidad Complutense de Madrid

Facultad de informática

Trabajo de Fin de Grado 2016-2017

Plataforma de aprendizaje de lenguas flexivas

Universidad Complutense de Madrid

Facultad de informática

Dedicado a todas esas personas que

nunca nos dieron por imposibles.

Agradecimientos:

A todo el profesorado que ha tenido que soportarnos estos años.

A todos los trabajadores de la UCM, especialmente a los de FDI, por hacernos estos años de trabajo y estudio más llevaderos.

Gracias, en particular, a Antonio Sarasa y a Ana Fernandez-Pampillon por acompañarnos y guiarnos en el último reto de la carrera.

ÍNDICE

1. RESUMEN

El proyecto de fin de grado que se presenta en esta memoria, ha consistido en el desarrollo de una aplicación e-learning para facilitar aprendizaje del latín. Esta herramienta aplica una metodología docente desarrollada en la Facultad de Filología de la Universidad Complutense de Madrid y validada, en el marco de un proyecto de innovación educativa de la UCM, en un centro de educación secundaria de la Comunidad de Madrid.

La aplicación permite que los alumnos puedan, si lo necesitan, complementar las clases presenciales o bien aprender de forma autoformativa. Incorpora un diccionario didáctico, desarrollado también en la Facultad de Filología y proporciona una interfaz gráfica de trabajo para ayudar a los estudiantes a entender el funcionamiento de la lengua latina mediante un símil de un rompecabezas. Asimismo permite el trabajo cooperativo de profesores, en la creación de actividades y de alumnos en su resolución. Para ello, la aplicación gestiona usuarios con el rol de profesor, para crear las actividades didácticas; el rol de alumno para resolverlas; y el rol de administrador, para gestionar las altas de nuevos profesores y alumnos, la base de datos que utiliza la aplicación y la modificación de los datos de los usuarios registrados.

Palabras clave: Aplicación e-learning, aprendizaje de lenguas, tecnología educativa.

1. ABSTRACT

The aim of this Degree's Final Project was to develop an e-learning application to facilitate the learning of Latin. This tool applies a teaching method developed by members of the Faculty of Philology of the "Universidad Complutense de Madrid", and validated in the framework of an educational innovation project of the UCM in a secondary school center of the Community of Madrid.

The application allows the students to complement their classroom lessons or "self-learning". It incorporates a dictionary, also developed in the Faculty of Philology, and provides a graphical working interface to help the students to understand how Latin language works through a simile of a puzzle. It also allows the cooperative work of teachers and students, in the creation of activities and their resolution, respectively. For this aim to be achieved, the application manages: users with the "teacher" role who create the didactic activities; users with the "student" role who solve those activities; and the "administrator" role who to manage the registration of new teachers and students, the data base used by the application and the modification of the data of registered users.

Keywords: E-learning application, languages learning, education technology.

2. PRESENTACIÓN DEL PROYECTO

2.1. Motivación

El aprendizaje del latín en educación Secundaria presenta importantes dificultades a los estudiantes entre otras razones porque los métodos de aprendizaje se basan en el uso de análisis sintácticos como una estrategia para la comprensión y generación de frases en latín y una parte importante de los alumnos no tienen suficientes conocimientos de sintaxis en los que apoyarse para aprender. Esto conduce a una falta de motivación que todavía dificulta más la enseñanza y el aprendizaje. Para abordar estos problemas, en la Facultad de Filología se diseñó y aplicó con buenos resultados un método didáctico nuevo basado en el uso de la semántica en vez de la sintaxis para la comprensión del latín en sus niveles iniciales [Márquez, M; Fernández Pampillón, A. (2016). El Diccionario Funcionalista como instrumento de autoaprendizaje de una lengua. Experiencias previas a la digitalización. Documento técnico disponible en Eprints UCM: http://eprints.ucm.es/38294/]. Asimismo, para facilitar la aplicación del método se creó un diccionario, primero en papel y después en formato electrónico (http://repositorios.fdi.ucm.es/DiccionarioDidacticoLatin/) en el marco de un proyecto de Innovación Educativa de la UCM (PIE 269-2016).

Persistía, sin embargo, el problema de que el aprendizaje sólo podía llevarse a cabo en las clases presenciales y con la presencia del profesor que era quien planteaba las actividades y las resolvía. El tiempo limitado de las clases presenciales no era suficiente para algunos alumnos, que necesitaban un poco más de dedicación. En este contexto es donde se plantea la necesidad de desarrollar una herramienta de aprendizaje electrónico, herramienta e-learning, accesible en la web, que permita a los profesores dejar planteadas las actividades autocorregibles y a los alumnos realizarlas a su propio ritmo y según sus necesidades de aprendizaje.

En este sentido, el objetivo principal del presente Trabajo de Fin de Grado ha sido desarrollar una aplicación e-learning que se ajuste al máximo a la metodología didáctica diseñada para el aprendizaje del latín. Esta metodología utiliza el símil de un rompecabezas para explicar el

funcionamiento de la lengua a nivel de oración. Básicamente, en una oración el verbo es la pieza que aglutina y determina el resto de los componentes tanto semántica como morfológicamente. Así, para entender (o traducir o generar) una oración primero se debe localizar el verbo y, en función de sus "valencias semánticas", se localizan el resto de los componentes (agente-nominativo, objeto-acusativo, etc) para que "encajen" con dichas valencias. Las piezas del rompecabezas (verbos, sustantivos, adjetivos, adverbios) se encuentran en un diccionario didáctico digital accesible en: http://repositorios.fdi.ucm.es/DiccionarioDidacticoLatin/.

2.2. Motivation

The learning of Latin in secondary schools often causes problems in students for several reasons. Learning methods are based on the use of syntantic analysis and an important part of the students don't have enough knowledge of this syntaxis to support their learning. This situation leads to a lack of motivation that enhances the teaching and learning problem. To face this problems, the Faculty of Philology designed and applied in the initial levels with good results a new didactic method based on the use of semantics instead of syntaxis [Márquez, M; Fernández Pampillón, A. (2016). El Diccionario Funcionalista como instrumento de autoaprendizaje de una lengua. Experiencias previas a la digitalización. Documento técnico disponible en Eprints UCM: http://eprints.ucm.es/38294/]. To also facilitate the method's application, a dictionary was created, first in paper and then in electronic format (http://repositorios.fdi.ucm.es/DiccionarioDidacticoLatin/), in the framework of an educational innovation project of the UCM (PIE 269-2016).

The main limitation was that the method was only available for its application during the classroom time, sometimes insufficient for some students. In this context, the need to develop an electronic learning (e-learning) tool that allows teachers to propose activities and students to solve them at their own pace, arises.

In this sense, the main objective of this Degree's Final Project was to develop an e-learning application that fits the didactic methodology designed for the learning of Latin. This methodology uses a simile of a puzzle to explain how the language works at sentence level. Basically, inside a sentence, the piece that agglutinate and determine both semantically and morphologically the rest of the components of the sentence is the verb. Thereby, to understand (or translate or generate) a sentence, you must first locate the verb, and on the basis of its "semantic valences", you can then locate the rest of the components (nominative-agent, accusative-object, etc) so as to they can "fit" with such valences. The pieces of the puzzle (verbs, nouns, adjectives, adverbs) can be found in a digital didactic dictionary accessible at: http://repositorios.fdi.ucm.es/DiccionarioDidacticoLatin/.

2.3. Objetivo

El objetivo del trabajo fin de grado era crear una aplicación web que implementase el método de aprendizaje de latín mencionado facilitando así la enseñanza y el aprendizaje.

Los objetivos específicos del proyecto han sido:

- Ofrecer una herramienta que facilitara al docente crear ejercicios y actividades en línea para el aprendizaje del latín basados en la metodología antes mencionada.
- Proporcionar la máxima flexibilidad (en cuanto a horarios y lugares) a los estudiantes para realizar actividades de aprendizaje del latín.
- Proporcionar una realimentación adecuada a los estudiantes sobre los ejercicios y actividades realizadas.
- Facilitar un acceso universal y simple a la herramienta que garantice que cualquier persona con conexión a internet pueda utilizarla.

- Proporcionar una interfaz usable para personas con conocimientos de informática a nivel de usuario.

2.4. Tecnologías utilizadas

La estructura de la aplicación está dividida en dos partes: frontend y backend.

La parte Front (cliente) está desarrollado en Angular2, con typescript como lenguaje de programación precompilado. Además, ha sido necesario el uso de otros frameworks fácilmente acoplables para extender la funcionalidad y dar solución a las necesidades del proyecto. Una de ellas ha sido la librería JQuery, una de las más conocidas y empleadas en proyectos webs para el Frontend. Para el manejo de colecciones, arrays y objetos, se ha utilizado la conocida librería de utilidades llamada Underscore.js.

Unos de los aspectos más importantes a la hora de realizar esta aplicación, es la importancia de que la forma en la que un alumno encajara las fichas del puzzle fuera lo más cómodo y dinámico posible. Es por ello que se optó desde el primer momento en usar una librería perteneciente a Angular2 que facilita la funcionalidad Drag&Drop (arrastrar y soltar) de elementos en la pantalla.

Para la vista, se ha utilizado el framework css conocido como Bootstrap, CSS3 y HTML5.

El servidor, desarrollado con Node.js, consiste en una Api Rest al cual se hacen llamadas desde el cliente para obtener todos los datos almacenados. Todos los datos son almacenados en una base de datos no relacional, MongoDb.

A continuación, se comentan las tecnologías mencionadas:

- **Angular2 [1]**

  Es un Framework para el desarrollo de aplicaciones web de código abierto con lenguaje typescrypt y mantenido por Google, utilizado para la realización de páginas web de una sola

página (SPA). Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador, para que el desarrollo y las pruebas sean más fáciles.

Es el sucesor de AngularJS, aunque es incompatible con este.

Angular2 es la solución a los problemas que empezaron a detectarse en su versión principal, el cual no cubría ciertas necesidades que se han solucionado en esta nueva versión.

Uno de los problemas principales era el uso de javascript, un lenguaje al que resulta muy difícil adaptarse. Con Angular 2 es posible hacer uso de ECMAscript 6, lo que mejora notablemente la legibilidad del código. Además, es posible hacer uso de transpiladores como Babel, el cual nos permite usar lenguajes como TypeScript, del que hablaremos en el siguiente punto.

Con AngularJS, la mayoría de la programación es trasladada al navegador, lo que provoca una sobrecarga de datos que puede ser perjudicial en aplicaciones con una carga de datos considerable. Además, AngularJS carga todo el código de la aplicación en la primera visita, lo que puede generar un impacto negativo.

Angular2 ha solucionado este problema, con Lazy SPA (Dependency injection), un inyector de dependencia que hace que no sea necesario descargar y conocer todo el código al acceder por primera vez, sino que que solicitando datos a medida que se van necesitando.

La arquitectura de una aplicación Angular ahora se realiza mediante componentes. En este caso no se trata de una novedad de la versión 2, ya que en la versión de Angular 1.5 ya se introdujo el desarrollo basado en componentes.

Sin embargo, la componentización no es algo opcional como en Angular 1.5, sino es una obligatoriedad. Los componentes son estancos, no se comunican con el padre a no ser que se haga explícitamente por medio de los mecanismos disponibles, etc. Todo esto genera aplicaciones más mantenibles, donde se encapsula mejor la funcionalidad y cuyo funcionamiento es más previsible. Ahora se evita el acceso universal a cualquier cosa desde cualquier parte del código, vía herencia o cosas como el "Root Scope", que permitía en

versiones anteriores de Angular modificar cualquier cosa de la aplicación desde cualquier sitio.

- **Typescript [15]**

  Es un lenguaje de programación libre creado y mantenido por Microsoft. Es un superconjunto de JavaScript que esencialmente añade tipado estático y programación orientada a objetos. Agrega las posibilidades de ES6 y el futuro ES7 y ayudas durante la escritura del código. Puede ser usado para el desarrollo de aplicaciones web que se ejecutarán en el lado del cliente o del servidor (Node.Js).

  TypeScript extiende la sintaxis de JavaScript y está pensado para grandes proyectos los cuales a través de un compilador de este lenguaje se traducen a JavaScript original.

- **JQuery [8]**

  Es una plataforma multiplataforma de JavaScript que permite simplificar la manera de interactuar con los documentos HTML.

  Es software libre y de código abierto. A igual que otras bibliotecas ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían emplear más líneas de código.

- **Underscore.js[16]**

  Es una biblioteca de JavaScript que proporciona funciones y utilidades para tareas de programación comunes, simplificando mucho las tareas a la hora de manejar arrays y objetos.

  Opta por un diseño de programación funcional en vez de extender los prototipos de objetos.

- **Librería drag&drop[10]**

  Drag and Drop (arrastrar y soltar) es un mecanismo basado en eventos, el API de JavaScript y elementos de marcado adicionales para indicar aquellos elementos de una página que pueden ser arrastrados.

Para este proyecto ha sido muy importante el uso de una librería drag&drop para la realización de la parte gráfica de los ejercicios como también poder hacer uso de ella en la ordenación de ejercicios, siendo más cómodo poder arrastrar un ejercicio a la posición deseada y hacer cambios en cualquier momento.

La librería escogida ha sido ng2-drag-drop de Angular2, por su sencillez, y por aportar una información clara de uso en la web de npm (repositorio de librerías).

- **Bootstrap[2]**

  Es un conjunto de herramientas de código abierto para diseño de aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, menús de navegación y otros elementos de diseño basados en HTML Y CSS.

  Basado en HTML y CSS así como extensiones JavaScript opcionales adicionales. Está considerado el proyecto más popular de GitHub y es usado por la NASA y la MSNBC entre otras organizaciones.

- **CSS3[3]**

  CSS (Cascading Stylesheets) u hojas de estilo en cascada, es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de las páginas web.

  Está diseñado principalmente para marcar la separación del contenido del documento y la presentación de este.

- **HTML5[6]**

  Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código para la definición de contenido de una página web.

  Basa su filosofía de desarrollo en la diferenciación (Para añadir un elemento externo no se incrusta en el código de la página si no que se hace referencia a su ubicación) recayendo sobre el navegador la tarea de unir ambos elementos y su visualización final.

- **Node.js[14]**

  Es un entorno en tiempo de ejecución multiplataforma, de código abierto para la capa del servidor. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables como servidores web. Al contrario que la mayoría del código JavaScript, no se ejecuta en navegador, sino en el servidor.

  Como en este caso puede ser combinado con una base de datos documental(otro ejemplo es CouchDB) y JSON lo que permite desarrollar en un entorno de desarrollo JavaScript unificado. Facilita la reutilización de código del mismo modelo de interfaz entre el lado del cliente y el servidor.

- **Mongodb[13]**

  Es un sistema de base de datos NoSQL orientado a documentos desarrollado bajo el concepto de código abierto.

  En lugar de guardar los datos en tablas como se hace en bases de datos relacionales, MongoDB guarda estructuras de datos en documentos similares a JSON con un esquema dinámico, haciendo que la integración de datos sea más fácil y rápida.

3. ESTADO DEL ARTE

Ante la búsqueda de aplicaciones similares en el ámbito de aprendizaje de latín se han encontrado algunas como lingQ[11] o linguim[12]. Estas aplicaciones usan sistemas clásicos de aprendizaje y evaluación, por ejemplo, LingQ usa un sistema a base de ejercicios donde al alumno se le presenta un texto en latín el cual puede reproducir por voz y se le proponen ejercicios referentes a dicho texto, como completar frases con la palabra indicada o la traducción de palabras aparecidas en el texto base del ejercicio, haciendo una corrección automática según la respuesta del estudiante. Por otra parte, linguim basa el aprendizaje en lecciones donde los ejemplos y su explicación teórica son el pilar fundamental del aprendizaje, pero no cuenta con ejercicios para practicar lo aprendido en la lección. Además, dichas aplicaciones no cuentan con la evaluación de los profesores que proponen las actividades a realizar pues su evaluación es directamente automática sin la supervisión final de un profesor que revise y comente las soluciones para un mayor entendimiento de las mismas.

En cuanto a herramientas de creación de ejercicios se encuentran algunas como JClic[7] orientada a la creación de actividades educativas como pueden ser rompecabezas, asociaciones, ejercicios de texto etc. Como en nuestra aplicación las actividades no se presentan individualmente sino en paquetes (equiparando a las actividades formadas por ejercicios de este proyecto fin de grado).

Otra herramienta es Hot Potatoes[5] , una herramienta creada por la Centro de Humanidades de la Universidad de Victoria (UVIC), en Canadá. Consta de una serie de programas para la creación de ejercicios interactivos multimedia. Cuenta con una serie de esquemas predeterminados en el que el autor introduce tanto las preguntas como las respuestas, así como los planteamientos de los ejercicios.

ExeLearning [4] también ayuda a los docentes a la creación de contenidos didácticos en soportes informáticos sin necesidad de un profundo conocimiento sobre informática. Con los recursos generados con Exelearning pueden generarse sitios web completos, incluir contenidos interactivos (preguntas y actividades de diferentes tipos), exportar los contenidos creados en diferentes formatos y catalogar el contenido con diferentes modelos de metadatos.

Por ultimo learningApps.org [9] es una es una aplicación creada para apoyar los procesos de aprendizaje y enseñanza con pequeños módulos. El objetivo es reunir módulos reutilizables y ponerlos a disposición de todos. Debido a esto los módulos no se ponen en un escenario específicos, sino que deben se consideran como unidades que deben ser integrados en la situación de aprendizaje adecuada.

Tras el análisis de estas aplicaciones se concluyó que no eran adecuadas para dar soporte a la metodología didáctica que motiva este trabajo.

4. REQUISITOS DEL SISTEMA.

A continuación, se describen los requisitos planteados para la realización de la aplicación.

- El sistema debe implementarse como una aplicación web accesible desde cualquier ordenador conectado a internet y sin requerir ningún software específico.
- Debe estar protegido contra intrusiones o ataques que puedan modificar el contenido de la web.
- La aplicación gestionará tres tipos de usuarios: estudiantes, docentes y administrador.
- Todo usuario docente o estudiante deberá registrarse antes de poder utilizar la aplicación.
- Debe registrar toda la información generada por los usuarios de la página en una base de datos (en este caso no relacional).

- El administrador del sistema se encargará tanto de la revisión como de la gestión de las peticiones de registro, del diccionario y de los datos guardados en la base de datos.
- El administrador podrá actualizar el diccionario desde un fichero xls.
- El administrador podrá subir guías de utilización para profesores y alumnos.
- La aplicación ofrece al profesor las herramientas necesarias para la gestión de creación de ejercicios y actividades en diferentes niveles con el apoyo de un diccionario online proporcionado por el cliente y alojado en un servidor externo al de la aplicación. Dicho diccionario cumplirá con un estándar ISO desarrollado por el cliente.
- El sistema asegura la integridad de la información. El alumno no podrá modificar/entregar una solución una vez mandada para evaluación o pasada la fecha de entrega de la actividad.
- El portal debe ofrecer un tiempo de respuesta adecuado incluso con multitud de conexiones simultáneas.
- El sistema permitirá a un estudiante abandonar una actividad y volver a retomarla en el punto donde la abandonó.
- El sistema permitirá a un estudiante solucionar parcialmente una actividad, sin la necesidad de realizar todos los ejercicios.
- El sistema permite a un alumno abandonar una actividad sin guardar el progreso.
- Se permitirá a los alumnos realizar las actividades todas las veces que se desee generando soluciones nuevas.
- El sistema permite a los alumnos hacer búsquedas filtradas de actividades y soluciones.
- El sistema permite a un profesor crear actividades sin fecha límite de entrega.

- El sistema permite a los profesores hacer búsquedas filtradas de ejercicios y actividades.
- El sistema permite a un alumno consultar el diccionario proporcionado para la resolución de los ejercicios que componen las actividades propuestas.
- El sistema permite al alumno usar la metodología de aprendizaje para resolver los ejercicios que componen las actividades.
- El sistema permite tanto a alumnos como a profesores consultar las guías de uso proporcionadas por el administrador para sus roles.
- El sistema permite a los profesores modificar o borrar los ejercicios creados por el mismo.
- El sistema permite a los profesores modificar o borrar las actividades creadas por el mismo.
- El sistema permitirá a un profesor utilizar los ejercicios de otro profesor para crear actividades.
- El sistema permitirá a un profesor proponer actividades con una fecha tope de entrega.
- El sistema permitirá a un profesor comentar y evaluar las soluciones propuestas de los alumnos de las actividades creadas por él.
- El sistema permitirá a un alumno revisar sus actividades resueltas después de ser evaluadas por el profesor.
- El sistema le permite al profesor crear actividades a partir de actividades ya existentes creadas por él.
- El sistema hace una primera evaluación, aproximada, de las soluciones dadas por los alumnos.

5. ANÁLISIS DEL SISTEMA

En este apartado, se describe el análisis funcional del sistema, el cual se va a describir mediante casos de uso y diagramas de actividades.

Actores del sistema

Los perfiles existentes en el entorno están definidos a continuación.

- **Administrador:** Es el encargado de mantener el sistema y de las altas y bajas de profesores y alumnos, gestión de usuarios y administración de guías y diccionario.
- **Profesor**: El rol de profesor es el encargado de crear tanto ejercicios como actividades, proponer las actividades, y corregir y comentar las respuestas de los alumnos.
- **Alumno**: Los alumnos podrán resolver los ejercicios de las actividades, obteniendo una valoración parcial por cada ejercicio y una global a nivel de actividad además de ver la corrección de sus actividades por parte del profesor una vez este las haya corregido.

Diagrama de casos de uso para cada actor

Seguidamente se reflejan los casos de uso de cada actor existente en la aplicación para una primera visión clara de las acciones que puede realizar cada uno.

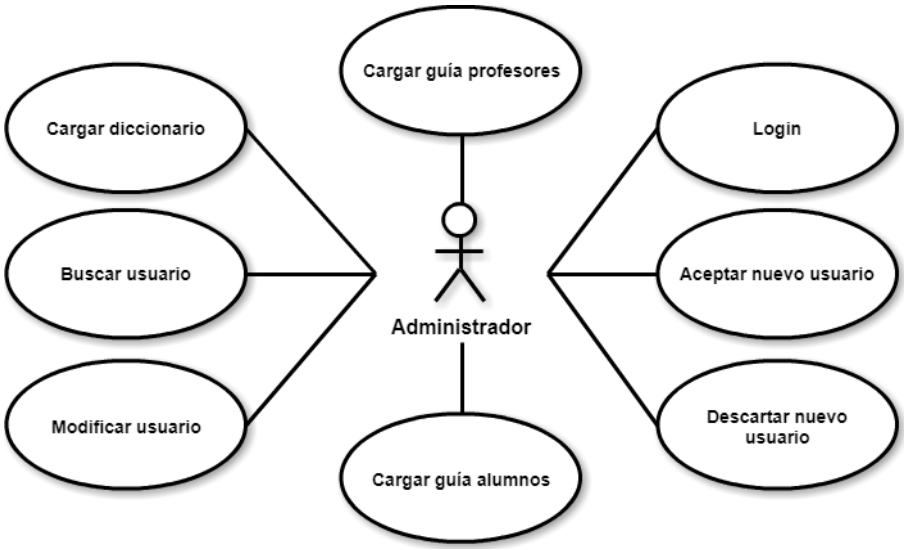


Figura 1. Diagrama de caso de uso de Administrador

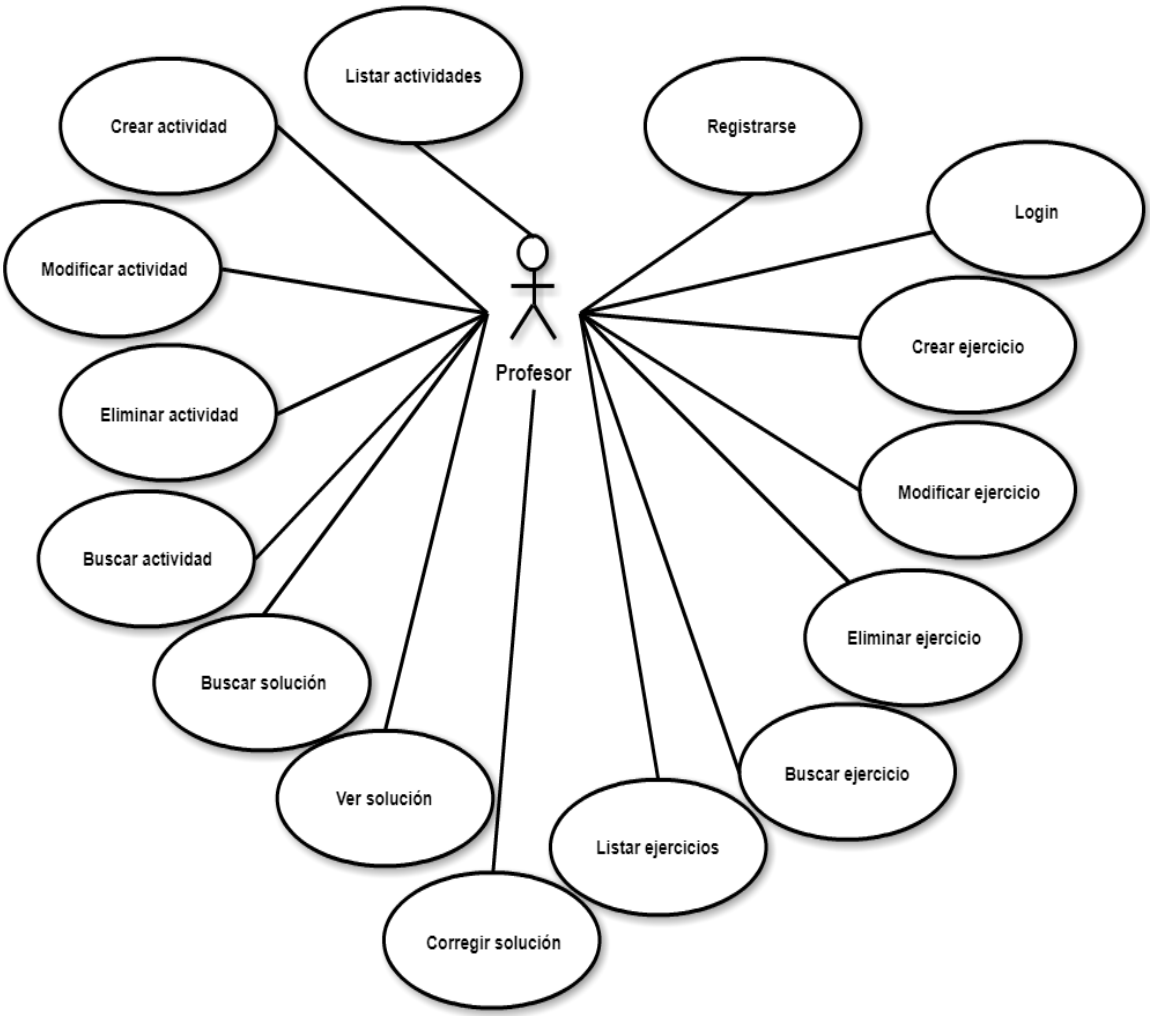


Figura 2. Diagrama de caso de uso de Profesor

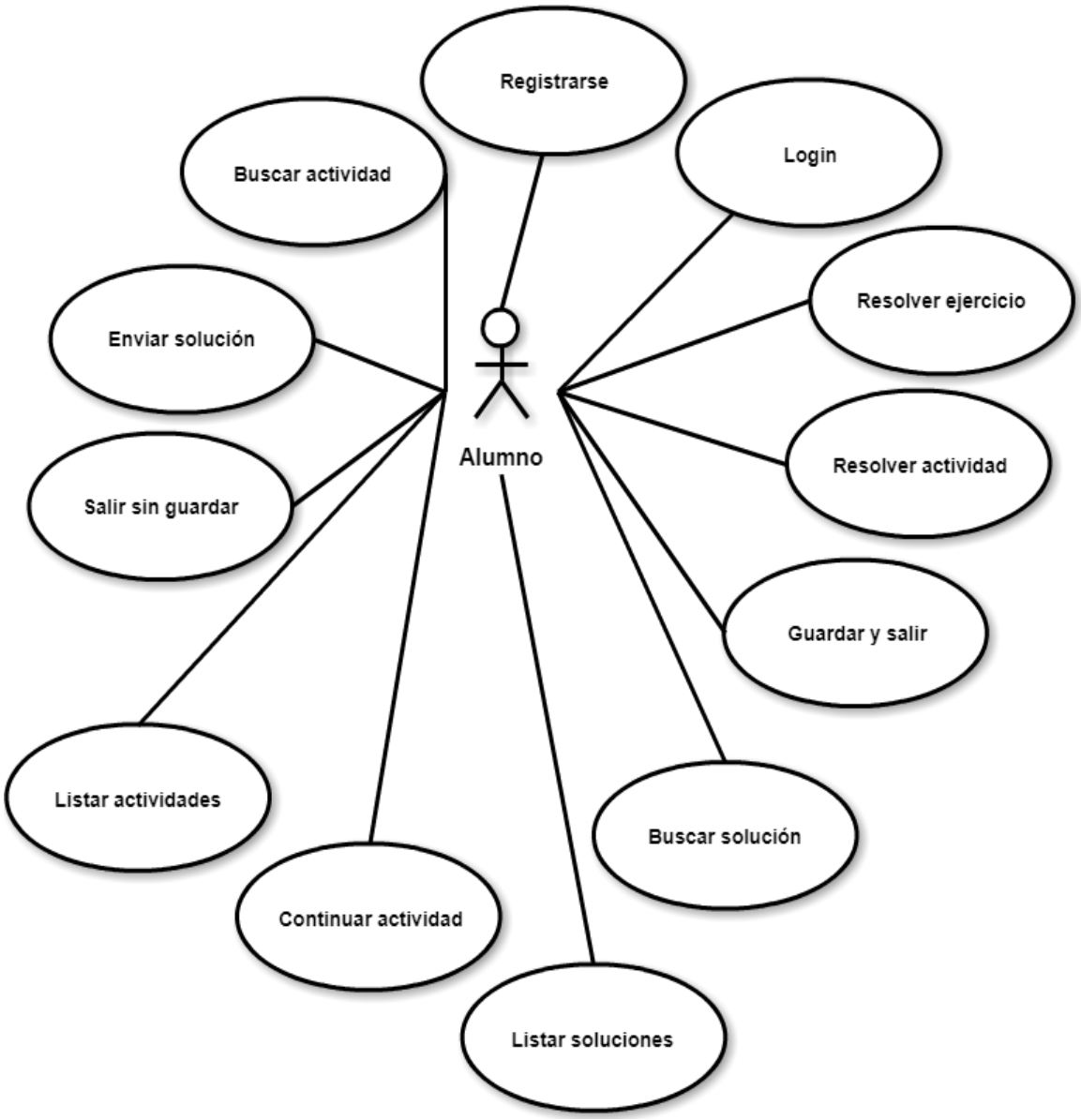


Figura 3. Diagrama de caso de uso de Alumno

5.1.Registro de usuarios

La pantalla inicial es un login con la opción de registro para nuevos usuarios. Ahí se solicitan los datos necesarios, incluyendo el tipo de rol que se solicita ser.

	REQUISITO 1: Registro de nuevo usuario
Prioridad	Alta.
Objetivo en contexto	Formulario para que un nuevo usuario introduzca sus datos de registro.
Descripción	Registrar un nuevo usuario en el portal web.
Entrada	Datos del usuario: Usuario, Contraseña, Nombre, Apellidos, Email, Fecha de nacimiento, Institución educativa y el perfil que se solicita (Profesor o Alumno).
Salida	Mensaje informativo.
Origen	Usuario.
Destino	Base de datos.
Necesita	Datos del usuario.
Acción	Crear.
Precondición	Datos correctos del usuario y que no esté ya de alta en la BBDD.
Post condición	Éxito: Si el administrador lo aprueba, se crea un perfil con los datos introducidos, y se da permiso de acceso al usuario.
Actores	Profesor/Alumno.

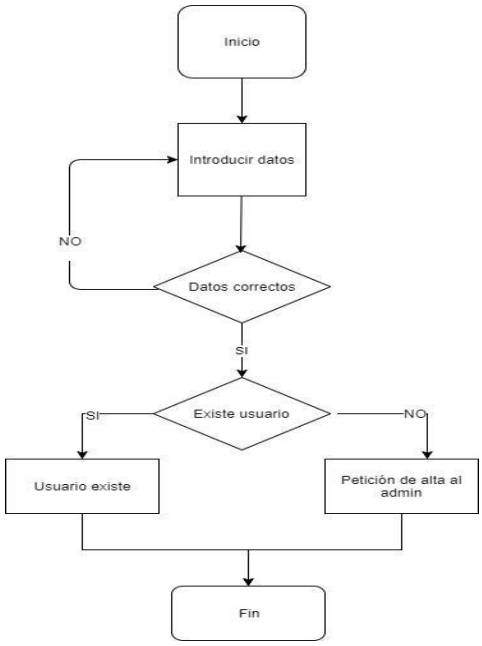


Figura 4. Registro de usuarios.

5.2.Login

Para poder acceder a su área privada, tanto el Administrador como los alumnos y los profesores, deberán introducir sus credenciales de usuario (nombre de usuario y password).

	REQUISITO 2: Login
Prioridad	Alta.
Objetivo en contexto	Formulario donde los miembros del portal corroboran su nombre de usuario y contraseña para poder acceder.
Descripción	Permitir el acceso al área personal de cada usuario.
Entrada	Datos de acceso del usuario: usuario y contraseña.
Salida	Pantalla principal del área personal (varía según el tipo de usuario).
Origen	Usuario
Destino	Base de datos
Necesita	Datos de acceso del usuario
Acción	Acceso
Precondición	Usuarios dados de alta en el portal.
Post condición	Éxito: Redireccionamiento a la página de área personal. Fallo: Mensaje informativo sobre la negación del acceso.
Actores	Administrador/Profesor/Alumno

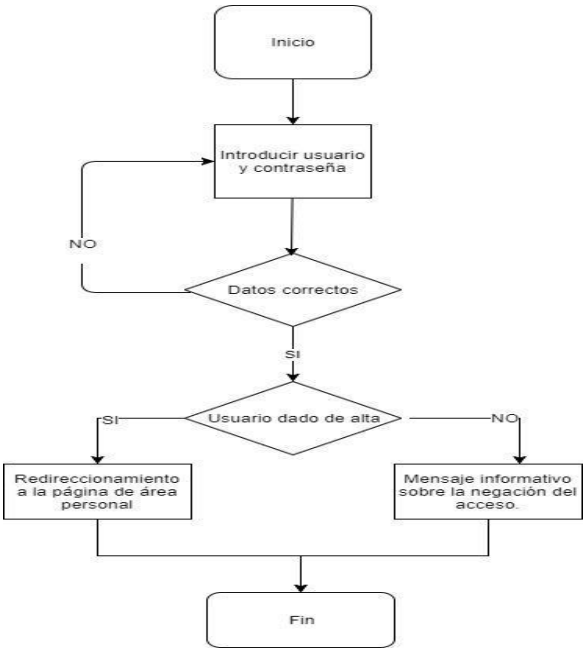


Figura 5. Login.

5.3.Aceptar nuevo usuario

Cada vez que un nuevo usuario haga una solicitud de registro, está tendrá que ser aceptada por el administrador una vez verifique los datos del nuevo usuario.

	REQUISITO 3: Aceptar nuevo usuario
Prioridad	Alta
Objetivo en contexto	Botón para aceptar que un usuario se registre si sus datos son correctos.
Descripción	Permitir que un nuevo usuario pueda acceder al portal web con los permisos del rol solicitado (profesor/alumno).
Entrada	Datos de registro del usuario.
Salida	
Origen	Formulario de registro.
Destino	Base de datos.
Necesita	Formulario de registro.
Acción	Dar de alta un nuevo usuario.
Precondición	Algún usuario haya solicitado registrarse.
Post condición	Éxito: El usuario es dado de alta en el portal.
Actores	Administrador

Figura 6. Aceptar nuevo usuario.

5.4.Descartar nuevo usuario

Si el administrador considera que los datos proporcionados por un usuario al solicitar ser dado de alta son erróneos, este podrá rechazar dicha solicitud.

	REQUISITO 4: Descartar nuevo usuario
Prioridad	Alta
Objetivo en contexto	Botón para impedir que un usuario se registre si sus datos no son correctos.
Descripción	Denegar que un nuevo usuario pueda acceder al portal web con los permisos del rol solicitado (profesor/alumno).
Entrada	
Salida	
Origen	Base de datos.
Destino	Base de datos.
Necesita	Formulario de registro.
Acción	No permitir un alta de un nuevo usuario.
Precondición	Algún usuario haya solicitado registrarse.
Post condición	Se borra de la bbdd la solicitud de registro del usuario.
Actores	Administrador

Figura 7. Descartar nuevo usuario.

5.5.Buscar nuevo usuario

El administrador podrá buscar a los usuarios registrados en la aplicación mediante el nombre de usuario.

	REQUISITO 5: Buscar usuario
Prioridad	Alta
Objetivo en contexto	Buscar un usuario que esté dado de alta.
Descripción	Función para encontrar usuarios dados de alta en la plataforma web.
Entrada	Nombre del usuario a buscar.
Salida	Lista de usuarios que cumplen los criterios de búsqueda.
Origen	Entrada por teclado.
Destino	Base de datos.
Necesita	Nombre de usuario.
Acción	Busca en la BBDD los nombres de usuario que coinciden con el insertado en el campo de búsqueda.
Precondición	Hay al menos un usuario registrado en la BBDD.
Post condición	Se muestra un listado con las coincidencias.
Actores	Administrador

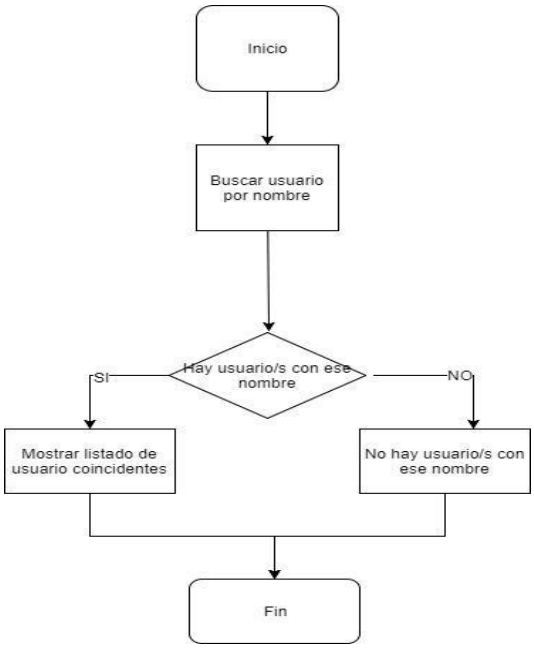


Figura 8. Buscar nuevo usuario.

5.6.Modificar usuario

El administrador podrá modificar los datos de los usuarios, tanto profesores como alumnos.

REQUISITO 6: Modificar usuario	
Prioridad	Alta
Objetivo en contexto	Formulario con los datos actuales de un usuario con los campos modificables activados.
Descripción	El administrador podrá modificar los datos de cualquier usuario.
Entrada	Datos a modificar del usuario.
Salida	Mensaje de confirmación de cambios.
Origen	Formulario de datos.
Destino	Base de datos.
Necesita	Formulario de datos actuales de un usuario existente.
Acción	Cambiar los datos de un usuario.
Precondición	El usuario existe.
Post condición	Se actualizan los datos del usuario con las modificaciones realizadas por el administrador.
Actores	Administrador

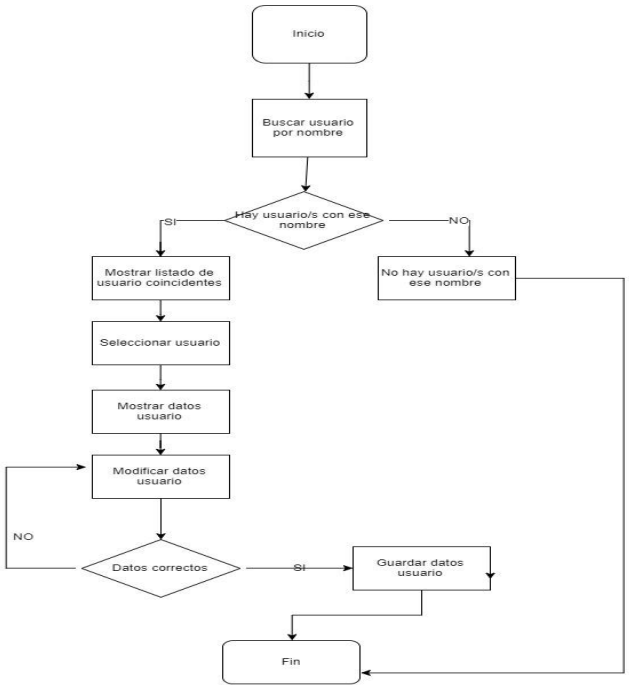


Figura 9. Modificar usuario.

5.7.Eliminar usuario

El administrador puede eliminar cualquier usuario que tenga acceso a la aplicación, independientemente del rol que tenga.

	REQUISITO 7: Eliminar usuario
Prioridad	Alta
Objetivo en contexto	Botón sobre cada usuario del listado que nos permite borrarlo de la BBDD.
Descripción	El administrador podrá eliminar un usuario de la BBDD.
Entrada	Datos a modificar del usuario.
Salida	Mensaje de confirmación de cambios.
Origen	Base de datos.
Destino	Base de datos.
Necesita	Usuario existente en la base de datos.
Acción	Elimina el usuario.
Precondición	El usuario existe.
Post condición	El usuario queda eliminado de la base de datos y ya no tiene acceso a la aplicación.
Actores	Administrador.

Figura 10. Eliminar usuario.

5.8.Cargar guía de profesores

Cualquier usuario con rol de administrador puede cargar desde su panel, una guía en formato .pdf al servidor, para que los profesores tengan acceso a la documentación de uso desde su panel.

	REQUISITO 8: Cargar guía de profesores
Prioridad	Alta
Objetivo en contexto	Formulario donde el usuario podrá seleccionar un archivo desde su equipo local para subir al servidor.
Descripción	El administrador podrá seleccionar un archivo en formato .pdf desde su equipo y subirlo al servidor para que los profesores tengan acceso a una guía de uso.
Entrada	Navegador de directorios para seleccionar archivo.
Salida	Mensaje de confirmación de que ha sido subido el archivo.
Origen	Usuario.
Destino	Interfaz web.
Necesita	Archivo s eleccionado en formato pdf.
Acción	Sube el archivo a un directorio de la aplicación.
Precondición	El archivo tiene que estar en formato pdf.
Post condición	El archivo es guardado en un directorio de la aplicación con un nombre específico. En caso de existir el archivo lo reemplaza.
Actores	Administrador

Figura 11. Cargar guía de profesores.

5.9.Cargar guía de alumnos

Cualquier usuario con rol de administrador puede cargar desde su panel, una guía en formato .pdf al servidor, para que los alumnos tengan acceso a la documentación de uso desde su panel.

	REQUISITO 9: Cargar guía de alumnos
Prioridad	Alta
Objetivo en contexto	Formulario donde el usuario podrá seleccionar un archivo desde su equipo local para subir al servidor.
Descripción	El administrador podrá seleccionar un archivo en formato .pdf desde su equipo y subirlo al servidor para que los alumnos tengan acceso a una guía de uso.
Entrada	Navegador de directorios para seleccionar archivo.
Salida	Mensaje de confirmación de que ha sido subido el archivo.
Origen	Usuario.
Destino	Interfaz web.
Necesita	Archivo seleccionado en formato pdf.
Acción	Sube el archivo a un directorio de la aplicación.
Precondición	El archivo tiene que estar en formato pdf.
Post condición	El archivo es guardado en un directorio de la aplicación con un nombre específico. En caso de existir el archivo lo reemplaza.
Actores	Administrador

Figura 12. Cargar guía de alumnos.

5.10. Cargar diccionario

EL diccionario que utiliza la aplicación para resolver ejercicios es precargado en la base de datos al arrancar el servidor. Dicho diccionario es un archivo en formato .xls. En cualquier momento, el administrador tiene la opción de volver a subir otra versión del diccionario, sobrescribiendo la que hubiese anteriormente.

	REQUISITO 10: Cargar diccionario
Prioridad	Alta
Objetivo en contexto	Formulario donde el usuario podrá seleccionar un archivo desde su equipo local para subir al servidor.
Descripción	El administrador podrá seleccionar un archivo en formato .xls desde su equipo y subirlo al servidor para actualizar la versión del diccionario.
Entrada	Navegador de directorios para seleccionar archivo.
Salida	Mensaje de confirmación de que ha sido subido el archivo.
Origen	Usuario.
Destino	Interfaz web.
Necesita	Archivo seleccionado en formato .xls.
Acción	Sube el archivo a un directorio de la aplicación.
Precondición	El archivo tiene que estar en formato .xls.
Post condición	El archivo es guardado en un directorio de la aplicación con un nombre específico. En caso de existir el archivo lo reemplaza.
Actores	Administrador

Figura 13. Cargar diccionario.

5.11. Crear ejercicio

Los ejercicios son creados por los profesores. La forma de crearlos es rellenar un formulario donde se especifica el título, el nivel de dificultad que estima el profesor, el enunciado, la frase a traducir, explicación del ejercicio, frase lematizada, la solución en formato lógico, en formato patrón y la posible solución en español.

	REQUISITO 11: Crear ejercicio
Prioridad	Alta
Objetivo en contexto	Formulario para definir un nuevo ejercicio.
Descripción	Crear un nuevo ejercicio en la base de datos.
Entrada	Datos del ejercicio que se desea crear.
Salida	Mensaje informativo.
Origen	Usuario.
Destino	Base de datos.
Necesita	Datos del ejercicio.
Acción	Alta nuevo ejercicio.
Precondición	El formulario de ejercicio ha sido rellenado.
Post condición	Éxito: Se da de alta un nuevo ejercicio. Fallo: Se informa al usuario del fallo.
Actores	Profesor.

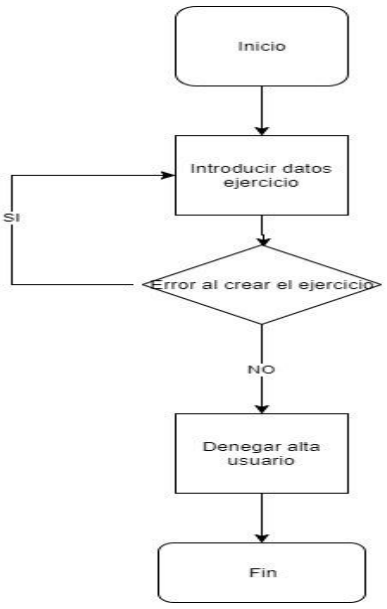


Figura 14. Crear ejercicio.

5.12. Modificar ejercicio

Función que permite a un profesor modificar algún dato de un ejercicio creado por él.

REQUISITO 12: Modificar ejercicio	
Prioridad	Media
Objetivo en contexto	Formulario con los datos del ejercicio a modificar, en el que se podrán cambiar los datos que se desee.
Descripción	Espacio donde el profesor podrá cambiar los datos de un ejercicio existente.
Entrada	Datos del ejercicio.
Salida	Datos del ejercicio modificado.
Origen	Usuario.
Destino	Base de datos.
Necesita	Datos del ejercicio a modificar.
Acción	Actualizar ejercicio.
Precondición	Ejercicio existente y creado por el usuario que quiere modificarlo.
Post condición	Ejercicio con los datos actualizados en la bbdd.
Actores	Profesor.

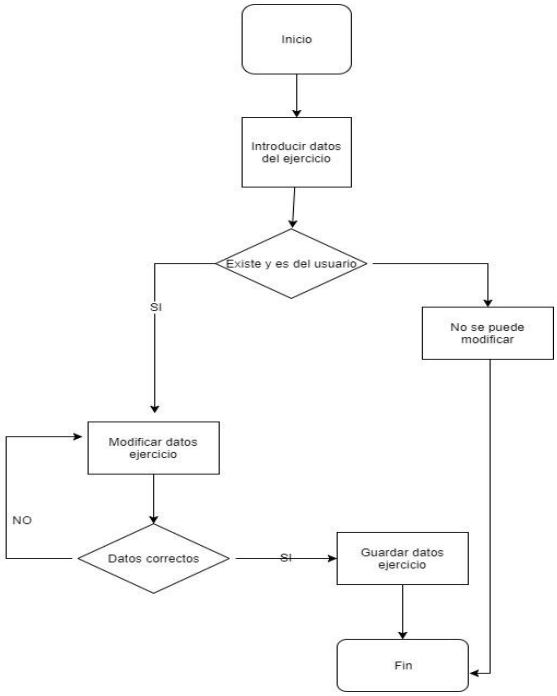


Figura 15. Modificar ejercicio.

5.13. Borrar ejercicio

Esta función permite ejecutar un borrado físico de un ejercicio en la base de datos.

REQUISITO 13: Borrar ejercicio	
Prioridad	Alta
Objetivo en contexto	Botón con mensaje de confirmación para borrar un ejercicio
Descripción	Borrado físico de un ejercicio
Entrada	Usuario
Salida	Confirmación del borrado.
Origen	Usuario.
Destino	Base de datos.
Necesita	Id ejercicio
Acción	Eliminar ejercicio.
Precondición	Existe el ejercicio y el profesor que lo quiere eliminar sea el que lo creó.
Post condición	Éxito: El ejercicio se borra de la bbdd y de todas las actividades y soluciones a las que esté asociado. Fallo: El ejercicio no se borra de la bbdd.
Actores	Profesor.

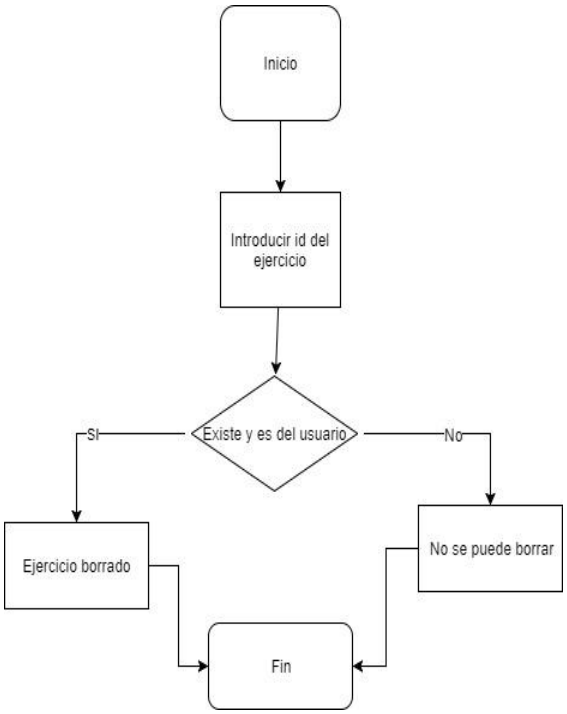


Figura 16. Borrar ejercicio.

5.14. Mostrar ejercicio

Los profesores podrán ver la información de interés de los ejercicios disponibles para poder decidir de una forma cómoda si quieren añadirlo a una actividad.

	REQUISITO 14: Mostrar ejercicio
Prioridad	Media
Objetivo en contexto	Botón que muestra una ventana modal con la información relevante de un ejercicio que se encuentre en la pantalla principal.
Descripción	Proporciona información de un ejercicio.
Entrada	Usuario
Salida	Datos del ejercicio.
Origen	Base de datos.
Destino	Interfaz gráfica.
Necesita	Id ejercicio.
Acción	Muestra ejercicio.
Precondición	Ejercicio existente y visible para el profesor solicitante.
Post condición	Pop-up con los datos del ejercicio.
Actores	Profesor.

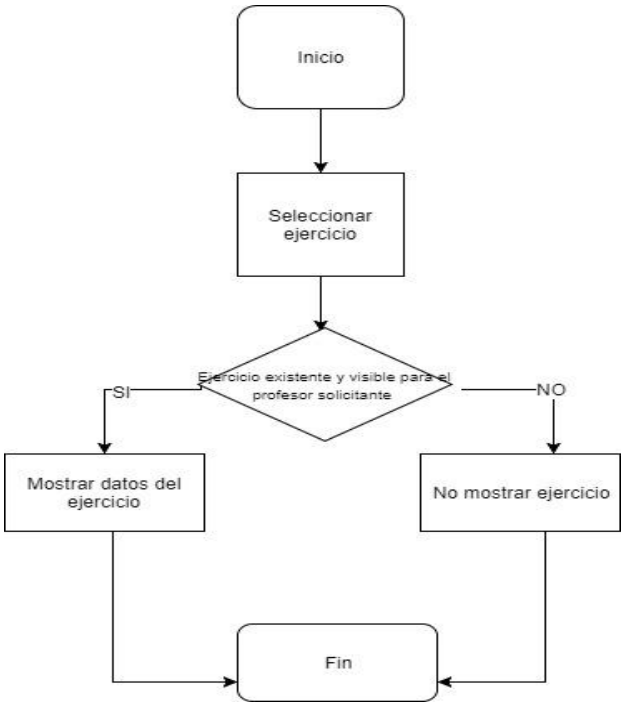


Figura 17. Mostrar ejercicio.

5.15. Buscar ejercicio

Con esta función se facilita encontrar ejercicios con facilidad a través de filtros.

REQUISITO 15: Buscar ejercicio	
Prioridad	Media
Objetivo en contexto	Se proporciona un filtro donde seleccionar los criterios de búsqueda (título, autor, frase a traducir, rango de fechas).
Descripción	Permite al profesor filtrar ejercicios
Entrada	Criterios de búsqueda.
Salida	Lista de ejercicios que cumplen las condiciones.
Origen	Usuario
Destino	Interfaz web
Necesita	Ejercicios creados
Acción	Búsqueda
Precondición	Existen ejercicios creados en la base de datos por parte de los profesores
Post condición	Éxito: El listado de ejercicios muestra los ejercicios que cumplan el filtro para ejercer las acciones deseadas sobre él. Fallo: Mensaje informando de que no se han encontrado resultados.
Actores	Profesor.

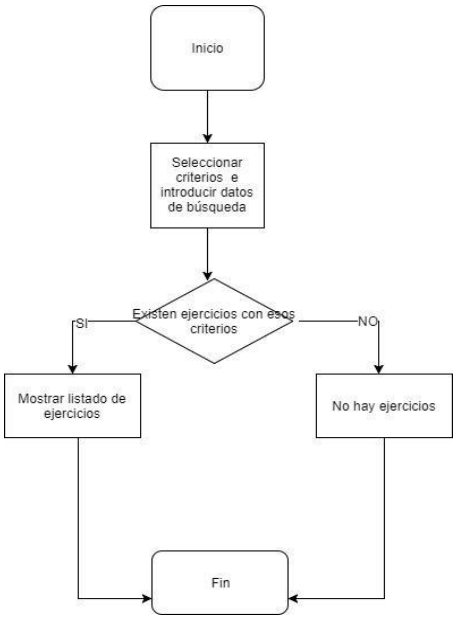


Figura 18. Buscar ejercicio.

5.16. Listar ejercicios

En la página principal de profesor hay un árbol desplegable a distintos niveles para poder ir filtrando el listado de ejercicios disponibles.

	REQUISITO 16: Listar ejercicios
Prioridad	Baja.
Objetivo en contexto	Se proporciona un desplegable donde poder listar los ejercicios existentes a diferentes niveles.
Descripción	Permite buscar una actividad específica filtrando por nombre.
Entrada	Nombre del ejercicio en cuestión.
Salida	Ejercicio solicitado en el listado de ejercicios.
Origen	Usuario
Destino	Interfaz web(listado de ejercicios)
Necesita	Ejercicios creados y nombre a buscar.
Acción	Búsqueda
Precondición	Existen ejercicios creados en la base de datos por parte de los profesores
Post condición	El listado de ejercicios al nivel deseado.
Actores	Profesor/Alumno.

Figura 19. Listar ejercicios.

5.17. Crear actividad

Se crean actividades añadiendo ejercicios en un panel disponible a la derecha de la pantalla. Tras añadir todos los ejercicios deseados, el sistema nos pedirá añadir los datos de dicha actividad, como son el título, la dificultad, si se quiere hacer visible y/o proponer la actividad con fecha de caducidad.

	REQUISITO 17: Crear actividad
Prioridad	Alta.
Objetivo en contexto	Crear actividades añadiendo los ejercicios y los atributos necesarios.
Descripción	Permitir la creación de actividades por parte del profesor con sus ejercicios y datos correspondientes.
Entrada	Ejercicios seleccionados y datos correspondientes.
Salida	Pantalla principal del área personal (varía según el tipo de usuario).
Origen	Usuario
Destino	Base de datos
Necesita	
Acción	Acceso
Precondición	Existen ejercicios con los que crear la actividad.
Post condición	Éxito: Se muestra mensaje de éxito en la creación de la actividad, ye esta se aloja en la base de datos. Fallo: Mensaje de error informativo.
Actores	Profesor.

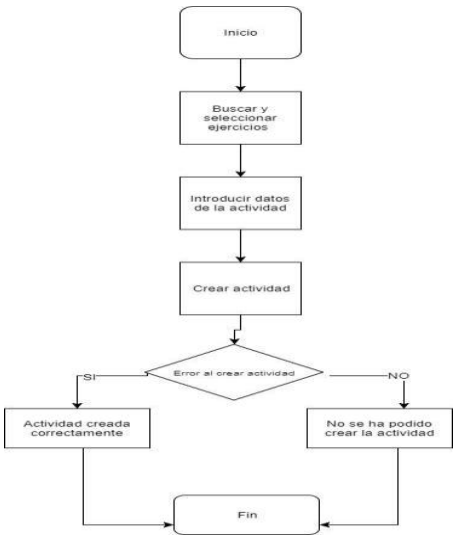


Figura 20. Crear actividad.

5.18. Buscar actividad

Con esta función se facilita encontrar actividades con facilidad a través de filtros.

REQUISITO 18: Buscar actividad	
Prioridad	Media
Objetivo en contexto	Se proporciona un filtro donde seleccionar los criterios de búsqueda.
Descripción	Permite al profesor filtrar actividades.
Entrada	Criterios de búsqueda.
Salida	Lista de actividades que cumplen las condiciones.
Origen	Usuario
Destino	Interfaz web
Necesita	Actividades visibles para el profesor o el alumno.
Acción	Búsqueda
Precondición	Existen actividades visibles en la base de datos para el profesor o el alumno.
Post condición	Éxito: El listado de actividades muestra las actividades que cumplan el filtro para ejercer las acciones deseadas sobre ella. Fallo: Mensaje informando de que no se han encontrado resultados.
Actores	Profesor/Alumno.

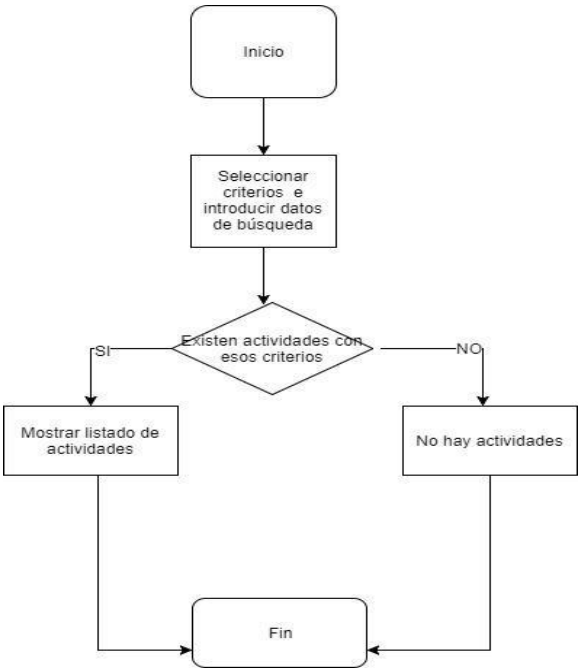


Figura 21. Buscar actividad.

5.19. Modificar actividad

Función que permite a un profesor modificar algún dato o añadir/quitar ejercicios de una actividad creada por él.

	REQUISITO 19: Modificar actividad
Prioridad	Media
Objetivo en contexto	Formulario con los datos de la actividad a modificar, en el que se podrán cambiar los datos que se desee.
Descripción	Espacio donde el profesor podrá cambiar los datos de una actividad existente.
Entrada	Datos de la actividad.
Salida	Datos de la actividad modificada.
Origen	Usuario.
Destino	Base de datos.
Necesita	Datos de la actividad a modificar.
Acción	Actualizar actividad.
Precondición	Actividad existente y creado por el usuario que quiere modificarlo.
Post condición	Actividad con los datos actualizados en la bbdd.
Actores	Profesor.

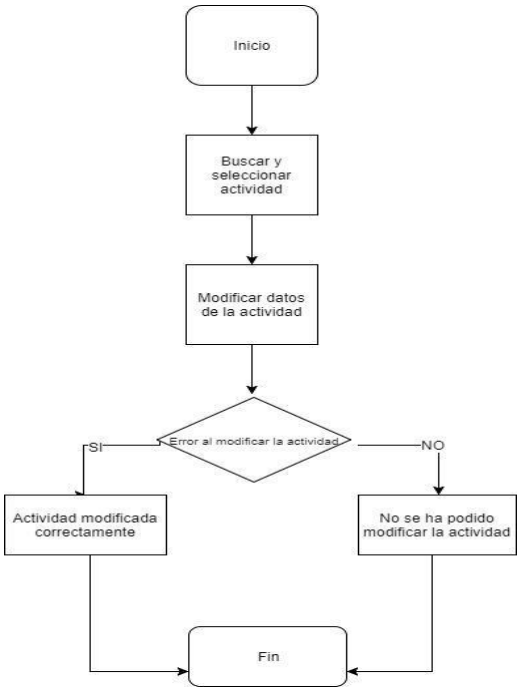


Figura 22. Modificar actividad.

5.20. Borrar actividad

Esta función permite ejecutar un borrado físico de una actividad en la base de datos.

REQUISITO 20: Borrar actividad	
Prioridad	Media.
Objetivo en contexto	Botón con mensaje de confirmación para borrar una actividad.
Descripción	Borrado físico de una actividad.
Entrada	Usuario
Salida	Confirmación del borrado.
Origen	Usuario.
Destino	Base de datos.
Necesita	Id actividad.
Acción	Eliminar actividad.
Precondición	Existe la actividad y el profesor que lo quiere eliminar sea el que la creó.
Post condición	Éxito: La actividad se borra de la bbdd. Fallo: La actividad no se borra de la bbdd.
Actores	Profesor.

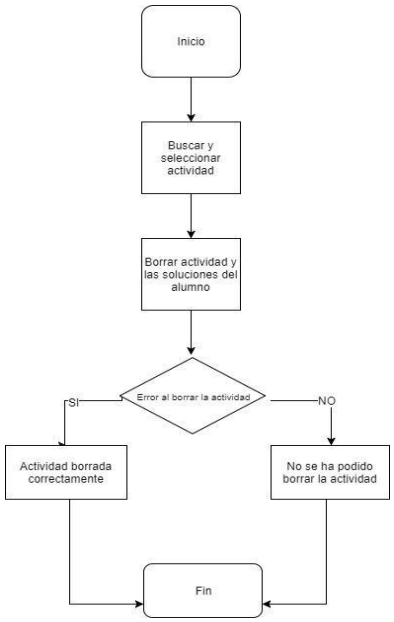


Figura 23. Borrar actividad.

5.21. Buscar solución

Con esta función se facilita encontrar soluciones con facilidad a través de filtros.

REQUISITO 21: Buscar solución	
Prioridad	Media
Objetivo en contexto	Se proporciona un filtro donde seleccionar los criterios de búsqueda.
Descripción	Permite a profesores y alumnos filtrar soluciones.
Entrada	Criterios de búsqueda.
Salida	Lista de soluciones que cumplen las condiciones.
Origen	Usuario
Destino	Interfaz web
Necesita	Profesor: Soluciones de actividades creadas por el profesor. Alumno: Soluciones de actividades realizadas por el alumno.
Acción	Búsqueda
Precondición	Existen soluciones en la base de datos pertenecientes al alumno o profesor.
Post condición	Éxito: El listado de soluciones muestra las soluciones que cumplan el filtro. Fallo: Mensaje informando de que no se han encontrado resultados.
Actores	Profesor/Alumno.

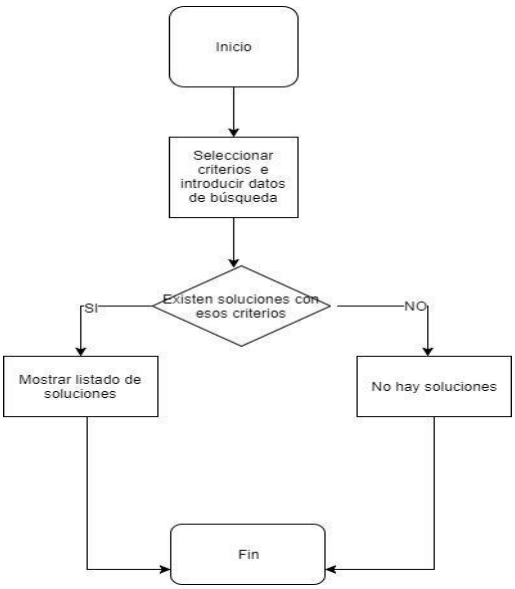


Figura 24. Buscar solución.

5.22. Mostrar/Comentar solución

Los profesores podrán ver la solución dada a sus actividades de una forma cómoda con la opción de modificar la nota obtenida y añadir comentarios a la solución dada para aclarar algún detalle al alumno que la realizó.

REQUISITO 22: Mostrar/Comentar Solución	
Prioridad	Media
Objetivo en contexto	Botón que muestra una ventana modal con la información relevante de una solución que aparezca después de aplicar el filtro de búsqueda.
Descripción	Información de una solución y deja añadir comentarios y la nota.
Entrada	Usuario
Salida	Datos de la solución.
Origen	Base de datos.
Destino	Interfaz gráfica.
Necesita	Id solución.
Acción	Muestra la solución y habilita el campo de comentarios.
Precondición	Solución existente de una actividad creada por el profesor.
Post condición	Ventana con los datos de la solución.
Actores	Profesor

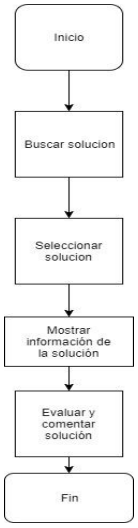


Figura 25. Mostrar/Comentar solución.

5.23. Resolver actividad

Esta función carga una actividad desde su inicio para que el alumno proponga su solución.

REQUISITO 23: Resolver actividad	
Prioridad	Alta.
Objetivo en contexto	Se podrá escoger la opción resolver actividad, redirigiendo al usuario a la pantalla de resolución de ejercicios.
Descripción	Permitir al alumno resolver los ejercicios que constituyen la actividad elegida.
Entrada	Solución a los ejercicios propuestos en la actividad
Salida	Pantalla de resolución del primer ejercicio de la actividad.
Origen	Usuario
Destino	Base de datos
Necesita	Actividades creadas y propuestas por el profesor
Acción	Acceso
Precondición	La actividad propuesta se encuentra dentro del plazo marcado por el profesor para su resolución.
Post condición	Éxito: Se muestra mensaje confirmando la resolución de la actividad, así como el correcto guardado de las soluciones obtenidas. Fallo: Mensaje informativo sobre la negación del acceso.
Actores	Alumno.

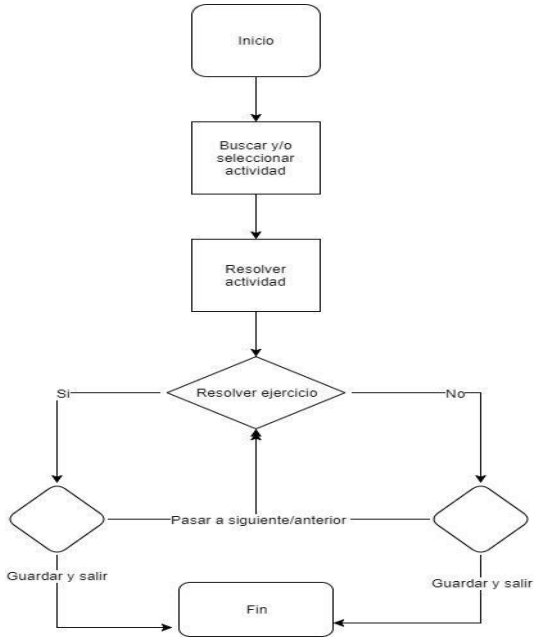


Figura 26. Resolver actividad.

5.24. Consultar diccionario

El alumno podrá abrir un pop-up con el diccionario a través del panel de resolución de actividad. En dicho pop-up, podrá hacer click sobre cualquiera de las palabras propuestas en dicho ejercicio y se le mostrará toda la información referente. No obstante, el alumno podrá buscar información de cualquier palabra que quiera.

REQUISITO 24: Consultar diccionario	
Prioridad	Alta.
Objetivo en contexto	En la pantalla de resolución de ejercicios se podrá seleccionar la opción diccionario mostrando el plugin con la información cargada para su consulta.
Descripción	El alumno tendrá acceso a la información del diccionario de latín una vez esté cargado en el plugin de la aplicación.
Entrada	Datos del diccionario alojado en el servidor oda.
Salida	Ventana o pop-up con los datos del diccionario cargado
Origen	Usuario
Destino	Plug-in de diccionario
Necesita	Diccionario alojado en un servidor externo a la aplicación
Acción	Acceso
Precondición	Usuarios dados de alta en el portal.
Post condición	Éxito: El alumno puede navegar y acceder a la información de dicho diccionario para su uso en la resolución de actividades. Fallo: Mensaje informativo sobre el error en la carga de los datos del diccionario al plugin de la aplicación.
Actores	Alumno

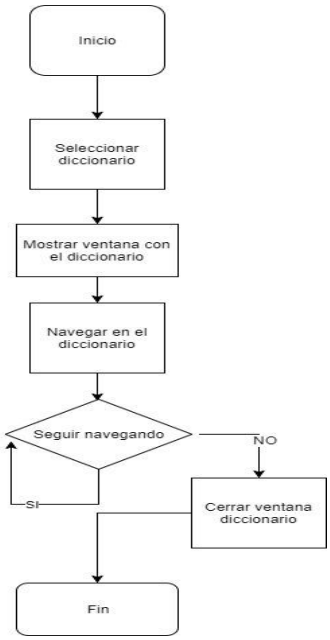


Figura 27. Consultar diccionario.

5.25. Mostrar actividad

Los profesores podrán ver la información de interés de las actividades disponibles.

REQUISITO 25: Mostrar actividad	
Prioridad	Alta.
Objetivo en contexto	En el listado de actividades, en cada una de ellas se encontrará la opción mostrar, que abrirá un popUp con los datos de la misma
Descripción	Permitir consultar los datos de una actividad en específico
Entrada	Actividad consultada.
Salida	PopUp con datos de la actividad
Origen	Usuario
Destino	Interfaz web
Necesita	Actividades creadas.
Acción	Acceso
Precondición	Existen actividades para consultar.
Post condición	Éxito: Se muestra popup con los datos de la actividad y los ejercicios que la componen. Fallo: Mensaje informativo sobre error al abrir el popUp.
Actores	Profesor.

Figura 28. Mostrar actividad.

5.26. Mostrar solución

Los alumnos podrán ver las soluciones de las actividades resueltas. Podrá consultar su nota global obtenida, la nota individual de cada ejercicio y su mensaje predefinido según nota, y los comentarios, si los hubiera, que haya dejado el profesor sobre cada uno de los ejercicios.

	REQUISITO 26: Mostrar solución
Prioridad	Media
Objetivo en contexto	Botón que muestra una ventana modal con la información relevante de una solución que aparezca después de aplicar el filtro de búsqueda.
Descripción	Muestra la solución del alumno junto con la nota y los posibles comentarios del profesor y las notas obtenidas.
Entrada	Solución a mostrar.
Salida	Información de la solución.
Origen	Base de datos.
Destino	Interfaz gráfica.
Necesita	Id solución.
Acción	Muestra la solución la nota y los comentarios.
Precondición	Solución existente y creada por el usuario solicitante.
Post condición	Ventana con los datos de la solución.
Actores	Alumno.

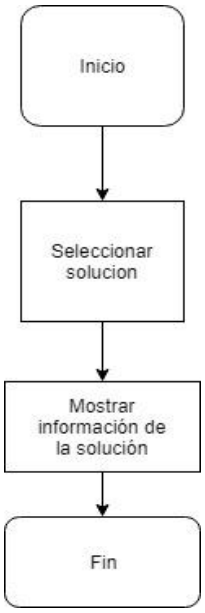


Figura 29. Mostrar solución.

5.27. Continuar actividad

En el panel principal de alumno, en la rama soluciones, hay un apartado: "Sin terminar". Ahí se dará la opción al alumno de continuar una actividad no terminada.

	REQUISITO 27: Continuar actividad
Prioridad	Alta.
Objetivo en contexto	Continuar una actividad empezada y no terminada.
Descripción	Los alumnos podrán retomar actividades que no hayan terminado por algún motivo.
Entrada	Solución inacabada.
Salida	Pantalla de resolución del ejercicio cargado en último lugar antes de abandonar la actividad.
Origen	Base de datos
Destino	Base de datos
Necesita	Id solución sin terminar.
Acción	Continuar actividad.
Precondición	Actividad existente, sin terminar y perteneciente al solicitante.
Post condición	Éxito: Se actualiza la solución con las nuevas modificaciones. Fallo: Mensaje informativo sobre la negación del acceso.
Actores	Alumno.

Figura 30. Continuar actividad.

5.28. Enviar solución

Una vez el alumno está satisfecho con las soluciones aportadas en una actividad podrá darla por finalizada y enviársela al profesor.

	REQUISITO 28: Enviar solución
Prioridad	Alta.
Objetivo en contexto	Guardar una solución y hacérsela visible al profesor.
Descripción	Los alumnos podrán terminar una actividad, guardar la solución y mandársela al profesor para una posible corrección.
Entrada	Solución terminada.
Salida	
Origen	Usuario.
Destino	Base de datos
Necesita	Orden de finalizar.
Acción	Guarda y envía la solución.
Precondición	Actividad finalizada.
Post condición	Éxito: Se guara la solución en bbdd y se hace visible para el profesor que creó la actividad. Fallo: Mensaje informativo del error.
Actores	Alumno.

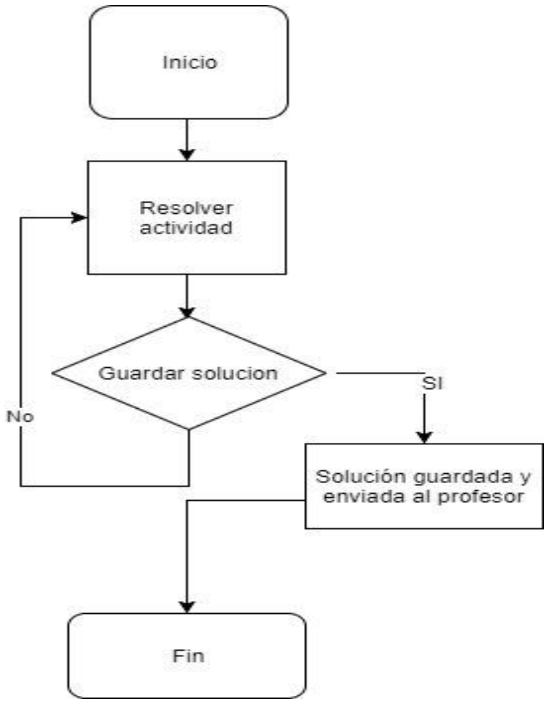


Figura 31. Enviar solución.

5.29. Guardar solución

Cada vez que el alumno termina un ejercicio de la actividad o navega por la actividad, se guarda automáticamente el progreso en background para no perder datos si ocurre algún problema.

REQUISITO 29: Guardar solución	
Prioridad	Alta.
Objetivo en contexto	Guardar una solución y hacérsela visible al profesor.
Descripción	Se guarda la solución de la actividad en cualquier estado y siempre en background cada vez que el alumno hace algún progreso.
Entrada	Progresos en la actividad.
Salida	
Origen	Usuario.
Destino	Base de datos
Necesita	Hacer cambios o navegar por la actividad.
Acción	Guarda el estado de la solución.
Precondición	El alumno hace cambios en la actividad.
Post condición	Éxito: Se guara la solución en bbdd.
Actores	Alumno.

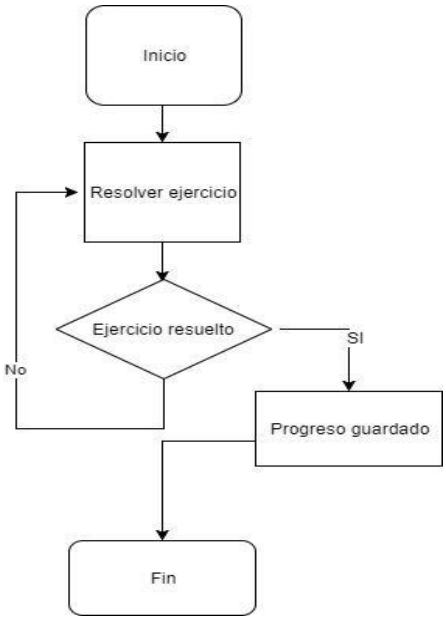


Figura 32. Guardar solución.

5.30. Salir

Funcionalidad que permite salir de una actividad sin guardar el progreso realizado hasta el momento. Para ello basta con pulsar sobre las migas de pan en “Panel principal”.

	REQUISITO 30: Salir
Prioridad	Media.
Objetivo en contexto	Abandonar una actividad sin guardar.
Descripción	Los alumnos podrán abandonar una actividad sin necesidad de guardar el progreso.
Entrada	Orden de salir.
Salida	Vuelta a la pantalla principal.
Origen	Usuario.
Destino	-
Necesita	Orden de salir.
Acción	Salir sin guardar
Precondición	Estar realizando una actividad.
Post condición	Se abandona la pantalla de resolución de actividad. Mensaje informativo del error.
Actores	Alumno.

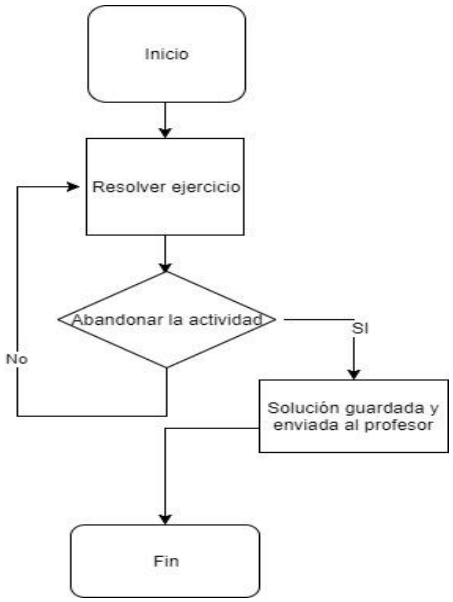


Figura 33. Salir.

5.31. Guardar y salir

Funcionalidad que permite salir de una actividad guardando el progreso realizado hasta el momento.

	REQUISITO 31: Guardar y salir
Prioridad	Media.
Objetivo en contexto	Guardar una solución sin terminar la actividad para poder retomarla más tarde.
Descripción	Los alumnos podrán abandonar una actividad sin perder el trabajo realizado en ella.
Entrada	Solución sin finalizar.
Salida	Mensaje informando del resultado de la función.
Origen	Usuario.
Destino	Base de datos
Necesita	Orden de salir y guardar.
Acción	Guarda la solución.
Precondición	Actividad sin finalizar.
Post condición	Éxito: Se guarda la solución en bbdd. Fallo: Mensaje informativo del error.
Actores	Alumno.

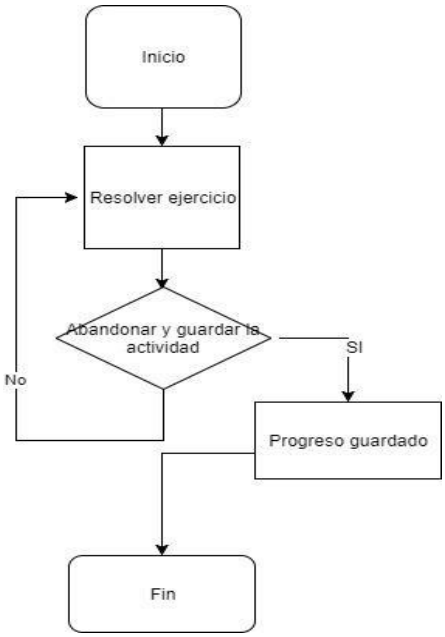


Figura 34. Guardar y salir.

5.32. Resolver ejercicio

Función con la cual el alumno puede resolver los ejercicios propuestos en una actividad.

	REQUISITO 32: Resolver ejercicio
Prioridad	Alta.
Objetivo en contexto	El alumno resuelve ejercicios.
Descripción	Los alumnos podrán resolver los ejercicios que conforman las actividades.
Entrada	Solución propuesta por el alumno para el ejercicio.
Salida	Nota del ejercicio
Origen	Usuario.
Destino	Base de datos.
Necesita	Id ejercicio, Id actividad, solución propuesta.
Acción	Crear solución de ejercicio.
Precondición	Id ejercicio perteneciente a la actividad en realización y orden de finalizar ejercicio.
Post condición	Se almacena la solución propuesta y se le da una nota provisional.
Actores	Alumno.

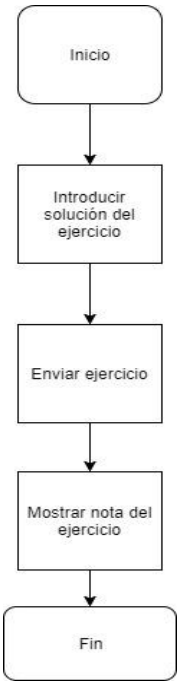


Figura 35. Resolver ejercicio.

5.33. Listar soluciones

En la página principal de alumno hay un árbol desplegable a distintos niveles para poder ir filtrando el listado de soluciones generadas por el usuario.

	REQUISITO 33: Listar soluciones
Prioridad	Baja.
Objetivo en contexto	Se proporciona un desplegable donde poder listar las soluciones generadas a diferentes niveles.
Descripción	Lista las soluciones, tanto finalizadas como no finalizadas, de un alumno.
Entrada	El nivel al que se quiere listar.
Salida	Lista de las soluciones al nivel especificado.
Origen	Usuario
Destino	Interfaz web(listado de soluciones)
Necesita	Soluciones generadas.
Acción	Listar
Precondición	Existen soluciones creadas en la base de datos por parte del alumno.
Post condición	El listado de soluciones las soluciones del alumno al nivel deseado.
Actores	Alumno

Figura 36. Listar soluciones.

6. ARQUITECTURA DE LA APLICACIÓN

La arquitectura de la aplicación sigue el patrón de diseño basado en componentes y el patrón modelo-vista-controlador.

El MVC es un patrón de desarrollo de software que busca separar la lógica del negocio y los datos de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

En la aplicación se pueden distinguir dos entidades principales, el cliente y el servidor. El cliente es el programa mediante el cual el usuario realiza peticiones a otro programa denominado servidor web el cual realiza tareas de procesamiento de la información y envía como resultado respuestas a las peticiones cursadas. Además, gestiona los datos y centraliza la administración de los mismos para lo que hace uso de un sistema de persistencia de información. En el caso particular de la aplicación realizada, la funcionalidad del servidor web se ha implementado utilizando Node.JS, el cliente lo constituye cualquier navegador web A través de Angular2 y la persistencia se implementa utilizando una base de datos NoSQL de tipo MongoDB.

Esta arquitectura presenta como ventaja la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. Además, la capacidad del proceso está repartida entre los clientes y los servidores.

En este caso, Angular2 utiliza componentes para la vista, enrutadores para la capa de control y servicios para la capa backend.

Un componente controla una zona de espacio de la pantalla que podríamos denominar vista. Cada componente lleva asociado un html, y toda la funcionalidad que queramos darle a la vista, modificando los datos y accediendo a los servicios.

Los servicios serán los encargados de conectarse con el servidor NodeJS para la extracción de datos, a través de los enrutadores que ofrece Angular2.

El modelo de datos está implementado en la parte servidor (NodeJS), donde a través de un framework para bases de datos MongoDB se hace el mapeo automático.

7. MODELO DE DATOS

En esta sección, se describe el modelo de datos usado. Para llevar a cabo la persistencia de la información se ha utilizado una base de datos no relacional de tipo MongoDB, en la cual se han definido las siguientes colecciones:

- **Colección actividad**: Colección que guarda las actividades creadas por un profesor.
  Al crear una actividad se guarda en esta colección que recoge los datos introducidos por el autor de la misma, así como los ids de los ejercicios que la componen en el atributo ejercicios para su futura recuperación.

Nombre	Formato	Descripción
Titulo	String	Nombre puesto por el profesor que crea la actividad
Id_profesor	String	Identificar único del profesor que crea la actividad
profesor	String	Nombre del profesor
Fecha_creacion	Date	Día/mes/año en la que se crea la actividad
nivel	String	Nivel de la actividad
ejercicios	Array ObjectId	Array de identificadores únicos de los ejercicios que componen la actividad
visible	Boolean	Campo que indica si la actividad es visible
propuesta	Boolean	Indica si la actividad está propuesta para su resolución
Fecha_prop_fin	Date	Día/mes/año máximo para la resolución de la actividad

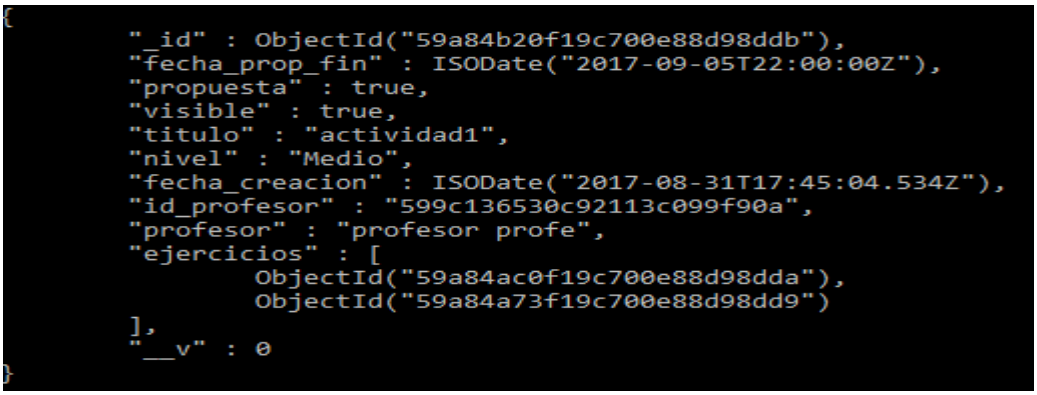


Figura 37. Colección actividades.

- **Colección ejercicios:** Colección que guarda los ejercicios creados por los profesores para la formación de actividades.

  Colección encargada de recoger los ejercicios creados. Tiene asociado el id del profesor autor del mismo para saber el propietario de dicho ejercicio y controlar su borrado o modificación e información útil para la resolución del ejercicio o la ayuda del diccionario (los atributos patrón y frase lematizada respectivamente).

Nombre	Formato	Descripción
Id_profesor	String	Identificador único del profesor
titulo	String	Título del ejercicio
nivel	String	Nivel del ejercicio
tipo	Number	Tipo del ejercicio
autor	String	Nombre del profesor
Intitucion_profesor	String	Institución a la que pertenece el profesor
fechaCreacion	Date	Día/mes/año de creación
fechaModificacion	Date	Día/mes/año de modificación
enunciado	String	Enunciado del ejercicio
descripcion	String	Descripción del ejercicio
FraseATraducir	String	Frase propuesta para el ejercicio
fraseLametizada	String	Frase lematizada
solucionFlogico	Date	solución en formato lógico
solucionFPatron	String	Patrón de la solución
solucionPEspañol	String	Solución en español
solucionPLatin	String	Solución en latín
explicacion	String	Explicación del ejercicio

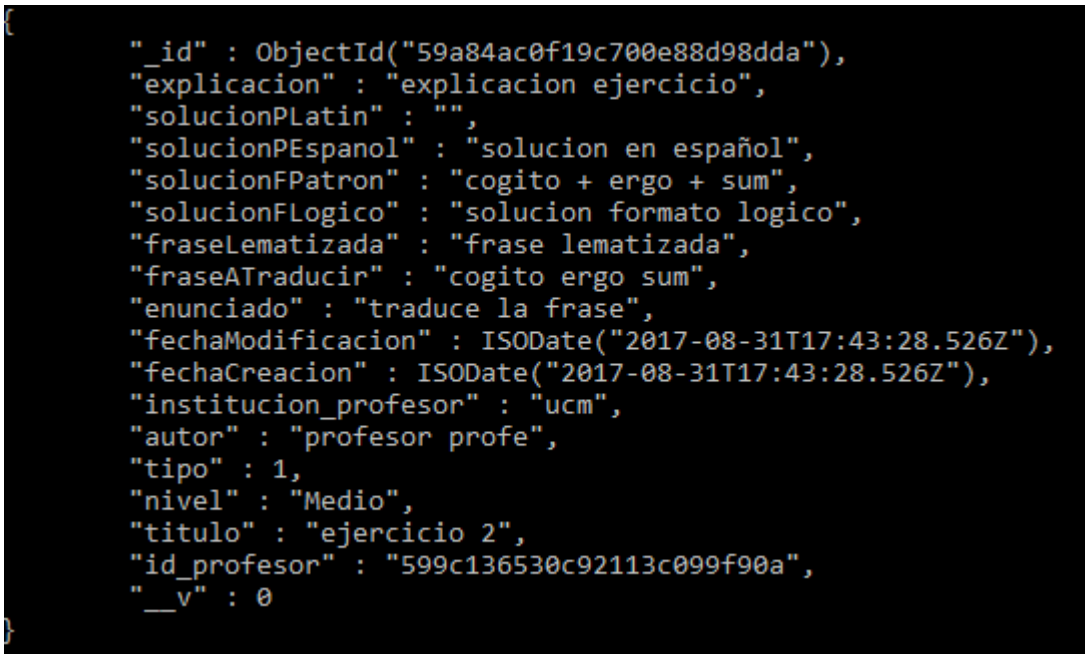


Figura 38. Colección ejercicios.

- **Colección registros:** Colección que guarda las peticiones de registro al sistema antes de ser aceptadas por el administrador.

  Está colección es la encargada de guardar las peticiones de registros de los usuarios a la aplicación. De ella se recupera el listado de peticiones que el administrador ve en su opción de aceptar registros. Una vez aceptados pasan a la colección users.

Nombre	Formato	Descripción
Usuario	String	Alias con el que se loguea el usuario
password	String	Contraseña de acceso a la aplicación
nombre	String	Nombre del usuario
apellidos	String	Apellidos del usuario
email	String	Email del usuario
Intitucion_educativa	String	Institución a la que pertenece el usuario
role	String	Rol del usuario(profesor)

```
{
        "_id" : ObjectId("599c136230c92113c099f908"),
        "role" : "alumno",
        "institucion_educativa" : "ucm",
        "email" : "luiszs080788@gmail.com",
        "password" : "$2a$10$wA.6wur9rjEOCNMW3N3UF.4r0H6jh91XLGTDqzFTev77WYyXJ/CgO",
        "apellidos" : "alum",
        "nombre" : "alumno",
        "usuario" : "alumno",
        "__v" : 0
}
```


Figura 39. Colección registros.

- **Colección users:** Colección que guarda los usuarios una vez han sido aceptados por el administrador

  En esta colección se guardan los usuarios aceptados en la aplicación. Con ella se controla el login al sistema y se recupera la información necesaria para las funcionalidades. Sobre esta colección es sobre la que actúa la opción de modificar datos de usuario del administrador por ejemplo.

Nombre	Formato	Descripción
Usuario	String	Alias con el que se loguea el usuario
password	String	Contraseña de acceso a la aplicación
nombre	String	Nombre del usuario
apellidos	String	Apellidos del usuario
email	String	Email del usuario
Intitucion_educativa	String	Institución a la que pertenece el usuario
role	String	Rol del usuario

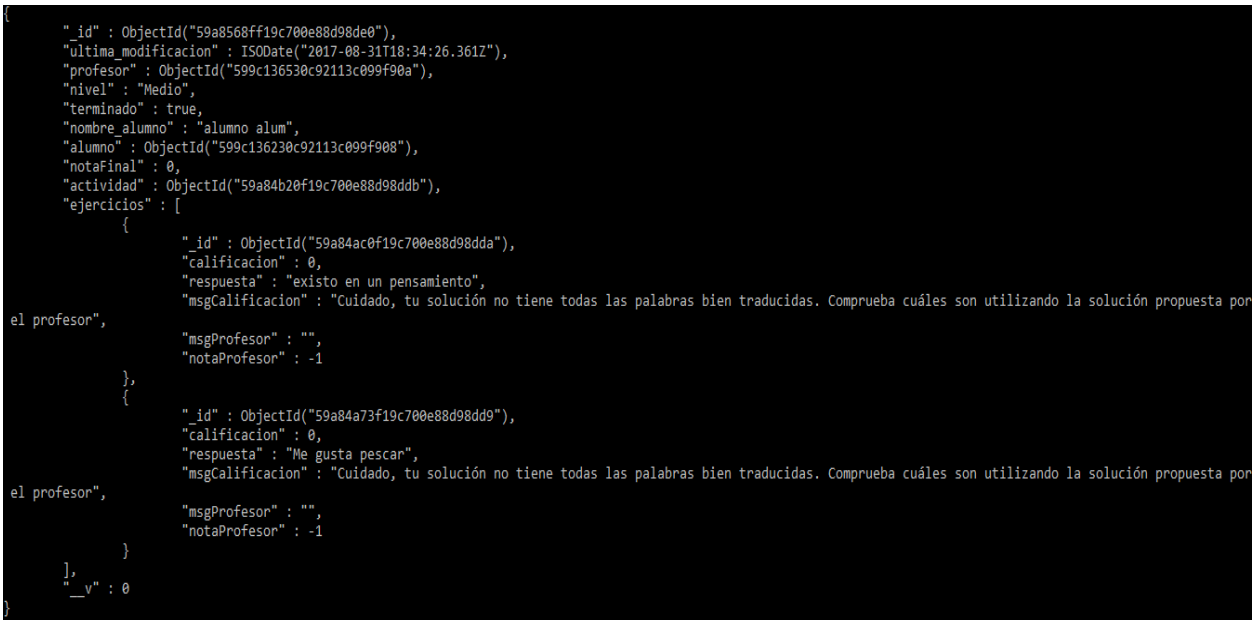
```
{
        "_id" : ObjectId("599c0ec630c92113c099d904"),
        "role" : "admin",
        "institucion_educativa" : "ucm",
        "email" : "luiszs080788@gmail.com",
        "password" : "$2a$10$2ZHD2eT66yYMzJROfV17M.xQrqPuOaYkdkMuIem1o.OfJFRhNgcHq",
        "apellidos" : "zorrilla",
        "nombre" : "luis",
        "usuario" : "admin",
        "__v" : 0
}
```

Figura 40. Colección users.

- **Colección soluciones:** Colección que guarda la solución de un creada por un alumno para una actividad propuesta

  Está colección recoge las soluciones enviadas por el alumno cuando resuelve una actividad. Contiene información necesaria para el correcto tratamiento de las soluciones, como el alumno propietario y los ejercicios que la componen, así como la actividad resuelta, su nota asociada etc.

Nombre	Formato	Descripción
actividad	ObjectId	Identificador único de la actividad resuelta
alumno	ObjectId	Identificador único del alumno que resuelve la actividad
Nombre_alumno	String	Nombre del alumno
nombre	String	Nombre del usuario
ejercicios	Array	Ids de los ejercicios que componen la actividad solucionada
notaFinal	Number	Nota obtenida en la actividad
nivel	String	Nivel de la actividad asociada
terminado	boolean	Indica si la actividad se ha terminado
profesor	ObjectId	Identificador único del profesor que propone la actividad
Ultima_modificacion	Date	Dia/Mes/Año de modificación



```
{
	"_id" : ObjectId("59a8568ff19c700e88d98de0"),
	"ultima_modificacion" : ISODate("2017-08-31T18:34:26.361Z"),
	"profesor" : ObjectId("599c136530c92113c099f90a"),
	"nivel" : "Medio",
	"terminado" : true,
	"nombre_alumno" : "alumno alum",
	"alumno" : ObjectId("599c136230c92113c099f908"),
	"notaFinal" : 0,
	"actividad" : ObjectId("59a84b20f19c700e88d98ddb"),
	"ejercicios" : [
		{
			"_id" : ObjectId("59a84ac0f19c700e88d98dda"),
			"calificacion" : 0,
			"respuesta" : "existo en un pensamiento",
			"msgCalificacion" : "Cuidado, tu solución no tiene todas las palabras bien traducidas. Comprueba cuáles son utilizando la solución propuesta por
el profesor",
			"msgProfesor" : "",
			"notaProfesor" : -1
		},
		{
			"_id" : ObjectId("59a84a73f19c700e88d98dd9"),
			"calificacion" : 0,
			"respuesta" : "Me gusta pescar",
			"msgCalificacion" : "Cuidado, tu solución no tiene todas las palabras bien traducidas. Comprueba cuáles son utilizando la solución propuesta por
el profesor",
			"msgProfesor" : "",
			"notaProfesor" : -1
		}
	],
	"__v" : 0
}
```


Figura 41. Colección soluciones.

- **Colección Diccionario:** Colección donde se carga el diccionario proporcionado por la facultad de filología para su consulta en la resolución de actividades.

Nombre	Formato	Descripción
GlobalInformation	Object	Información administrativa otros atributos generales de todo el recurso lexicón.
Lexicon	Object	Contiene todas las entradas léxicas.
LexicalEntry	Array de object	Entrada léxica, recoge todos los elementos de este tipo de entradas.
Lemma	String	Define el lema.
Sense	Array de object	Describe el marco predicativo.
PredicativeRepresentation	Array de object	Define el marco predicativo.
SemanticPredicate	Array de object	Expresa el orden de los argumentos.
Definition	String	Describe el significado del verbo.
SemanticArgument	Array de object	Argumentos semánticos.
fs	Array de object	rasgos semánticos.

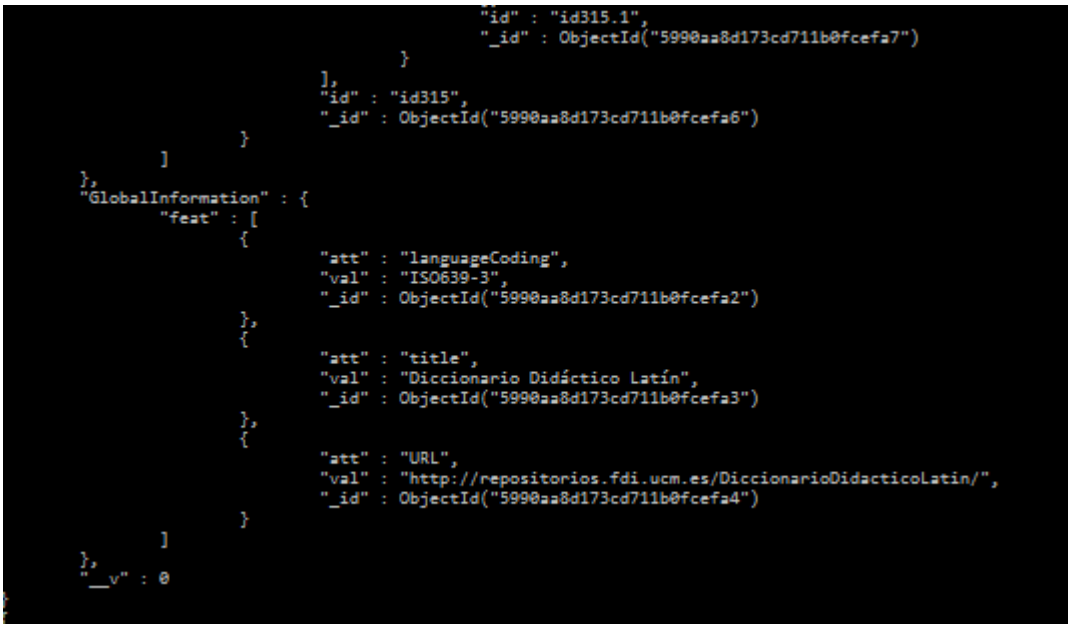


Figura 42. Colección lrs.

8. DISEÑO DEL SISTEMA

8.1. OBJETIVOS DEL DISEÑO

Como objetivo del diseño se marca facilitar al usuario su interacción con el sistema, haciendo que este no realice demasiadas tareas si no las esenciales para llevar a cabo su trabajo y hacer que el servidor lleve a cabo tareas de la gestión de datos creadas por el usuario como:

- Almacenar datos, así como la lógica de la aplicación.
- Controlar que no se dupliquen datos.
- Encargarse de la seguridad de la app

Antes de empezar cualquier proyecto, es necesario recopilar todos los requisitos que demanda el cliente. Es aquí donde se ha de tomar la decisión de qué estructura y lenguaje de programación se van a adaptar mejor al tipo de aplicación.

En nuestro caso, se trata de una aplicación con bastante administración de datos, pero que no requiere necesariamente hacer una división de pantallas con muchos menús y subniveles. Era una oportunidad perfecta para desarrollar una aplicación donde cada usuario, independientemente del rol que tenga, pueda manejar toda la información en una sola pantalla. Esto fue uno de los motivos que inclinó la balanza hacia el lado de Angular 2, ya que es una tecnología que permite desarrollar una web SPA (single page application), basada en componentes. Esto significa que podemos realizar y adaptar todas las necesidades que requiere, por ejemplo, un usuario con rol de profesor, en una sola página, mostrando las diferentes opciones que vaya requiriendo, empaquetadas en componentes.

Esta sería la parte correspondiente al cliente de nuestra aplicación. Para la parte servidor, se ha utilizado una tecnología que va muy bien de la mano de Angular 2, y que permite una comunicación asíncrona con este para acceder al modelo de datos. Estamos hablando de Node JS, junto con Express.

En las siguientes secciones se describe cada una de las partes.

8.2. Cliente

8.2.1. Componentes y navegación

Las aplicaciones en Angular 2 están basadas en componentes. El programador es libre de separar la aplicación en los componentes que crea necesario. Cada uno de estos componentes, son partes de una página web con funcionalidad propia. Se comunica con el servidor de forma asíncrona, esto quiere decir que no es necesario recargar la página para obtener datos. Cada componente lleva asociado su propio html y pueden comunicarse con otros componentes para transferir información

Un ejemplo de componente en esta aplicación web podría ser el buscador de soluciones, como se muestra en la siguiente imagen.

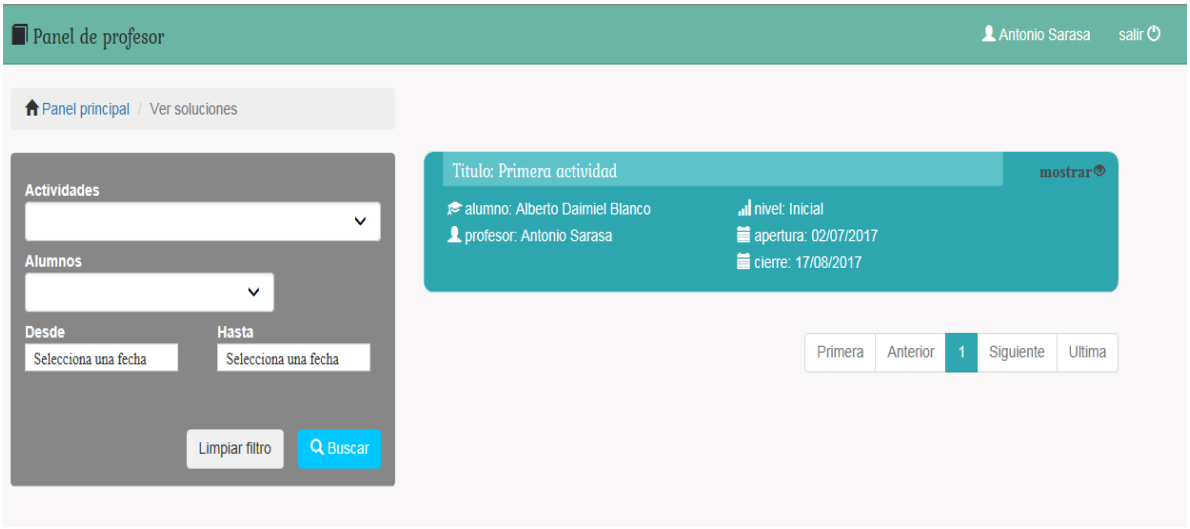


Figura 43. Buscador de soluciones

Como vemos, este componente forma parte de la vista de profesor, y es accesible a través del panel principal. Si quisiéramos volver atrás, no habríamos perdido ninguna información que hubiéramos gestionado en el primer panel, ya que esta ventana no es más que un componente que forma parte de la primera pantalla, que también es un componente.

Al no existir recarga de datos, todo lo que se gestione en cualquiera de las vistas disponibles, siempre quedará "guardado", permitiendo navegar entre las diferentes opciones sin perder información.

Ahora vamos a hablar de la navegación entre componentes. Esta aplicación está destinada para ser utilizada desde 3 roles diferentes: administrador, profesor y alumno.

Cada uno de estos 3 roles posee una página principal desde donde va a poder gestionar todas las opciones disponibles, siempre sin recargar la página. Por cada opción que requiere reordenar la información a mostrar en una nueva página, se mostrará siempre un camino de migas en la parte superior izquierda, para que el usuario pueda volver siempre al panel principal. En la imagen anterior, hemos accedido al buscador de soluciones a través del panel principal correspondiente al rol de profesor.

En la siguiente imagen, se muestra como ejemplo, la página principal de profesor desde la cual podemos gestionar todo.

Figura 44. Panel de profesor.

8.2.2. Diseño y usabilidad

Para llevar a cabo el diseño de la web, se decidió usar Bootstrap, un framework o conjunto de herramientas de código abierto para diseño de aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basados en HTML y CSS, así como extensiones de Javascript opcionales adicionales.

Algunos elementos que vienen por defecto en Bootstrap han sido alterados para mejorar la visualización por parte del usuario, como puedan ser botones, espacios entre elementos, colores, etc…

Además de esto, se han añadidos efectos visuales para que el usuario pueda percibir de una manera más visual las transiciones y acciones que ejerce sobre los elementos.

Otro de los puntos fuertes desde el punto de vista de la usabilidad, ha sido la incorporación de drag and drop en algunos elementos. Esto permite al usuario seleccionar algo en la página, arrastrarlo hasta otro punto y soltarlo. Esto es útil a la hora de reordenar una lista de ejercicios sobre una actividad, o para que el alumno pueda aprender latín mediante un sistema de fichas, realizando un puzle. A continuación, se muestra un ejemplo de ordenación de ejercicios y otro de un puzle.

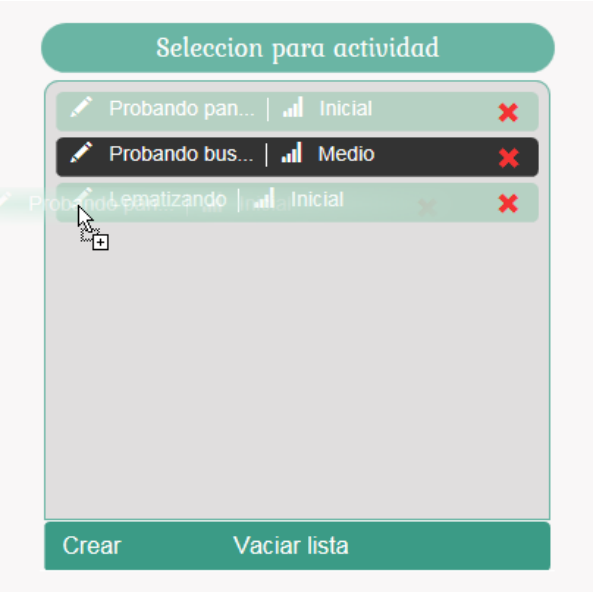


Figura 45. Ordenar ejercicios.

Figura 46. Encaje de fichas.

8.2.3. Comunicación con el servidor

Otra de las ventajas que ya hemos mencionado en Angular 2, es la posibilidad de interactuar con el servidor para obtener datos sin recargar la página, de manera asíncrona. Los que en otros lenguajes nos haría recurrir al uso de Ajax, con esta tecnología podemos hacer llamadas al servidor de una manera mucho más cómoda para el programador, que solo tiene que preocuparse de programas las funcionalidades de cada componente. Para ellos, se crean los servicios en la parte cliente, usados por los componentes. Estos servicios hacer llamadas Rest al servidor, y mapean la información devuelto en objetos que luego podremos leer en el componente.

Acabamos de mencionar que las llamadas van a ser de tipo Rest, por lo que ya podemos avanzar que el servidor va a consistir en montar una API Rest que se encargará de acceder al modelo de datos, entre otras cosas.

8.3. Servidor

En una arquitectura Angular-Node JS, todo el peso de negocio recae sobre la parte cliente. La parte servidor es la encargada de acceder a la base de datos, subir archivos a directorios e implementar un sistema de login basado en tokens, como ha sido en nuestro caso.

Para poder servir todos estos datos al cliente, es necesario montar una estructura de direcciones que todas juntas formarán nuestra API Rest.

Para ello, Node JS se apoya de un framework llamado Express, que, entre otras muchas opciones, nos permite montar un sistema de rutas para acceder desde cualquier cliente. En nuestro caso, hemos definido un sistema de rutas diferente para cada una de nuestras entidades en el modelo de datos. Cada una de estas rutas está vinculada con su función correspondiente.

Por otro lado, ha sido necesario utilizar otro framework llamado Mongoose para el mapeo de datos entre NodeJS y MongoDB. Mongoose aporta toda la funcionalidad CRUD necesaria en cualquier sistema de acceso a datos, además de ofrecer una gama muy amplia de funcionalidades que facilitan la gestión de los datos de una manera más cómoda.

Para poder gestionar las dependencias del servidor ha sido necesario el uso de NPM (Node Package Manager). Es un gestor de paquetes que nos permite obtener cualquier librería que necesitemos a través de la consola de comandos. Además, npm nos permite arrancar y parar nuestro servidor en cualquier momento.

8.4. Usabilidad de la página web

La web consta de tres tipos de usuarios cada uno encargado de una parte diferenciada e importante para el funcionamiento de la misma:

- Profesor: Creará ejercicios (así como podrá modificarlos o borrarlos) y con ellos creará actividades que los alumnos deberán resolver. Se encargará de la corrección de las actividades resueltas por los alumnos.

- Alumnos: Resolverá las actividades propuestas por los profesores apoyándose en el diccionario de latín proporcionado. Podrá dejar una actividad a medias retomándola más adelante en el estado que la dejó para su resolución.

- Administrador: Podrá aceptar o rechazar registros desde su página principal, modificar los datos de los usuarios, cambiar contraseñas y cargar tanto guías de usuario para alumnos y profesores como cargar diccionarios.

8.5. Fiabilidad

Se tendrá presente la consistencia de los datos y su integridad a la hora de la modificación o borrado de los mismos, acción de gran importancia al basarse la aplicación en un sistema de actividades compuestas por ejercicios que pueden ser modificados o borrados.

Se cuidará la validación CSS para que la aplicación funcione en diferentes navegadores, así como las validaciones HTML, ambos tratados como elementos fundamentales.

9. IMPLEMENTACIÓN

A continuación, se muestra la implementación realizada.

9.1. Implementación del cliente

La parte del cliente ha sido programada bajo la tecnología de Angular2 y para la comunicación se ha utilizado el API REST. El API es el encargado de devolvernos la información a través de objetos JSON. Para poder comunicarnos con el api, en el cliente se definen una serie de servicios que serán los encargados de hacer las peticiones de forma asíncrona, como si fuera a través de Ajax.

Para esta aplicación han sido necesarios los siguientes servicios:

1) **actividad.service.ts:** Aquí se implementan todos los métodos que llamarán a la parte del api que gestionan la entidad actividad. Por ello, es necesario definir cuál será la url base con la que se irán formando las diferentes direcciones según el método al que llamemos.

   this.url= 'http://'+window.location.hostname+':3678/api2/';

   Un ejemplo de llamada para obtener todas las actividades sería el siguiente.

```
getActividades(){
    return this._http.get(this.url+'actividades').map(res => res.json());
}
```

Figura 47. Servicio actividad.service.ts.

A través del objeto http de angular se ejecuta una llamada de tipo Get que obtendrá los datos en formato JSON y serán procesados en el componente que esté ejecutando este servicio.

Los demás servicios siguen la misma estructura, cambiando en cada uno de ellos la Url base que apunta a su correspondiente parte del Api en el servidor.

2) **authentication.service.ts:** Este servicio es el encargado de implementar los métodos que estén relacionados con la autenticación y el registro de usuarios, así como funciones sobre estos.

this.url= 'http://'+window.location.hostname+':3678/apiAuth/';

3) **diccionario.service.ts:** Este servicio únicamente sirve para hacer una extracción del diccionario completo al cargar el panel de alumno a través de una petición de tipo Get. En el componente se hará uso del objeto JSON devuelto para almacenarlo en una variable.

```
constructor(private _http: Http){
    this.url = 'http://'+window.location.hostname+':3678/apiDiccionario/';
}

getDiccionario(){
    return this._http.get(this.url+'getDiccionario').map(res => res.json());
}
```

Figura 48. Servicio diccionario.service.ts.

4) ejercicio.service.ts: Se encargará de implementar todos los métodos que manejen datos referentes a la entidad ejercicio, haciendo llamadas al api. Se muestra en la imagen un fragmento con la url del API con la que ha de comunicarse y un método que obtendrá ejercicios comprendidos entre dos fechas dadas

```
constructor(private _http: Http){
    this.url= 'http://'+window.location.hostname+':3678/api/';
}

getEjerciciosFecha(fecha1: Date, fecha2: Date){


    let params= {"fecha1":JSON.stringify(fecha1),
                "fecha2": JSON.stringify(fecha2)};

    let headers= new Headers({'Content-Type': 'application/json'});

    return this._http.post(this.url+'ejercicios', params, {headers: headers}).map(res=> res.json());
}
```

Figura 49. Servicio ejercicio.service.ts.

5) profesor.service.ts: Servicio que implementa métodos de obtención de datos sobre la entidad profesor.

```
constructor(private _http: Http){
    //this.url= 'http://localhost:3678/apiProfesor/';
    this.url = 'http://'+window.location.hostname+':3678/apiProfesor/';
}

getProfesores(){
    return this._http.get(this.url+'profesores').map(res => res.json());
}

getProfesor(_id: string){
    return this._http.get(this.url+'profesor/'+_id).map(res => res.json());
}
```

Figura 50. Servicio profesor.service.ts.

6) **solucion.service.ts:** La entidad solución es manejada desde este servicio al igual que las demás.

```
constructor(private _http: Http){
    this.url= 'http://'+window.location.hostname+':3678/apiSolucion/';
}

saveSolucion(solucion : Solucion){
    let json = JSON.stringify(solucion);
    let params= json;

    let headers= new Headers({'Content-Type': 'application/json'});

    return this._http.post(this.url+'solucion', params, {headers: headers}).map(res=> res.json());
}
```

Figura 51. Servicio solución.service.ts.

7) **user.servive.ts:** Este servicio solo se encarga de obtener la lista de usuarios disponibles en la base de datos. Después en los componentes se trata el objeto JSON para extraer información más sesgada.

```
    constructor(
        private http: Http,
        private authenticationService: AuthenticationService) {
            this.url= 'http://'+window.location.hostname+':3678/apiAuth/';
    }

    getUsers(): Observable<User[]> {
        // add authorization header with jwt token
        let headers = new Headers({ 'Authorization': 'Bearer ' + this.authenticationService.token });
        let options = new RequestOptions({ headers: headers });

        // get users from api
        return this.http.get(this.url+'apiUsers/users', options)
            .map((response: Response) => response.json());
    }
}
```

Figura 52. Servicio user.service.ts.

Después de conocer qué funcionalidad tienen los servicios de nuestra parte cliente, vamos a ver ahora qué componentes forman nuestra aplicación. Cada componente lleva asociado una vista html, y es en el componente donde se define la lógica que accederá a los datos del servidor y actualizará la vista de forma asíncrona.

8) **buscar-soluciones.component.ts:** Este componente, como su nombre indica, es el encargado de buscar y listas las soluciones de los alumnos. Lleva declarada su propia vista "panel-buscar-soluciones.html". Este HTML es un panel con el buscador y sus filtros, y un espacio reservado para listar las soluciones obtenidas.

   El componente implementa todos los métodos necesarios para obtener las soluciones. El método más importante es el de "buscar", el cual recoge todos los campos que haya rellenado el usuario para, a través del servicio de soluciones que hemos explicado anteriormente, devolvernos los datos de dicha búsqueda en un objeto JSON.

   Este componente se usa además como partes de otro componente. Ha sido reutilizado para el componente del panel de profesor y el componente del panel de alumno.

```
buscar(){
    var criteria= new CriteriaSolucion();
    if(this.busquedaByActividad != null){
        criteria.id_actividad= this.busquedaByActividad;
    }
    if(this.busquedaByAlumnos != null){
        criteria.ids_alumnos= this.busquedaByAlumnos;
    }
    if(this.fecha_desde != null){
        criteria.desde= this.fecha_desde;
    }
    if(this.fecha_hasta != null){
        criteria.hasta= this.fecha_hasta;
    }
    criteria.terminado=true;
    this._solucionService.getByCriteria(criteria).subscribe(
        result=>{
            this.soluciones= result.soluciones;
            if(!this.soluciones){
                alert('Error en el servidor');
            }else{
                if(this.soluciones.length > 0){
                    this.setPageSoluciones(1);
                    this.mostrarSoluciones=true;
                    this.msgBusqueda="";
                    this.boolMsgBusqueda=false;
                }else{
                    this.pagedSoluciones=[];
                    this.mostrarSoluciones=false;
                    this.msgBusqueda="No hay soluciones para este criterio de busqueda";
                    this.boolMsgBusqueda=true;
                }
            }
        },
        error=>{
             this.errorMessage= <any>error;
            if(this.errorMessage != null){
                console.log(this.errorMessage);
                alert('Error en la peticion de mi coleccion');
            }
        }
    );
}
```

Figura 53. Componente buscar-soluciones.component.ts.

9) **cabecera-admin.component.ts, cabecera-profesor.component.ts y cabecera-alumno.component.ts:** Son las cabeceras comunes de cada uno de los paneles de los 3 roles diferentes que tenemos. Sus componentes tan solo emplean como lógica la funcionalidad de poder hacer logout del sistema. El resto forma parte del aspecto visual de cada panel.

Figura 54. Componente cabecera-admin.component.ts.

10) **datos-solucion.component.ts:** Este componente se utiliza para cargar una solución elegida por el usuario. La vista asociada se encarga de mostrar los datos que un alumno ha aportado como solución a una actividad. Este componente es usado en la vista de profesores y de alumnos. Con el rol de profesor, además ofrece la posibilidad de agregar un comentario en cada ejercicio y modificar la nota pre asignada por el sistema. A continuación, se muestra una imagen de uno de los métodos del componente encargados de guardar la valoración de profesor para un ejercicio. En él se observa cómo se hace uso del servicio SoluciónService, el cual se comunicará con el API.

```
guardarValoracion(){

    this.solucion.ejercicios[this.indice].msgProfesor=this.valoracion;

    this._solucionService.updateSolucion(this.solucion).subscribe(

       result =>{
            this.editarVal=false;
        },
        error => {
            this.errorMessage= <any>error;

            if(this.errorMessage != null){
                alert(this.errorMessage);
            }
        }
    );
}
```

Figura 55. Componente datos-solucion.component.ts.

11) **ejercicio-add.component.ts:** Este panel se encarga de recoger los datos de creación de un ejercicio. Consta de un HTML con el formulario que recoge todos los campos, y los métodos del componente necesarios para gestionar los datos y poder guardarlos en la base de datos. Se muestra en la imagen el script encargado de hacer la llamada al servicio para guardar un ejercicio.

```
public onSubmit(){
    this._ejercicioService.addEjercicio(this.ejercicio).subscribe(
        response =>{
            if(response.ejercicio){
                this.ejercicio= response.ejercicio;
                this.modalEjercicio = true;
                setTimeout(() => this.visibleAnimate = true);
            }
        },
        error =>{
            this.errorMessage= <any>error;

            if(this.errorMessage != null){
                console.log(this.errorMessage);
                alert('Error en la peticion');
            }
        }
    );
}
```

Figura 56. Componente ejercicio-add.component.ts.

12) **login.component.ts:** Es el punto de partida para acceder a la aplicación. Además, permite a al usuario poder registrarse. Su HTML asociado contiene los dos formularios, el de login y el de registro. En el componente se encuentra los dos métodos principales, login y registro. A continuación, se muestra una imagen del método que permite al usuario loguearse. En él se puede comprobar que, si el usuario ha sido encontrado en la base de datos, se redirecciona al panel correspondiente al tipo de rol al que pertenezca.

```
login() {
    this.loading = true;
    this.authenticationService.login(this.modelLogin.usuario, this.modelLogin.password)
        .subscribe(result => {
            if (result === true) {
                this.user= JSON.parse(localStorage.getItem('currentUser')).user;
                if(this.user.role == 'admin'){
                    this.router.navigate(['/admin']);
                }
                else if(this.user.role == 'profesor')
                    this.router.navigate(['/profesor']);
                else if(this.user.role == 'alumno')
                    this.router.navigate(['/alumno']);
            } else {
                this.error = this.MS.LOGIN_INCORRECTO;
                this.loading = false;
            }
        });
}
```

Figura 57. Componente login.component.ts.

13) **panel-admin.component.ts:** Es el panel principal del usuario administrador. En él no se encuentra mucha lógica de negocio, es el encargado de administrar los subcomponentes que contiene (gestionar los nuevos registros, gestionar los usuarios de la aplicación, cargar las guías de usuarios y el diccionario de la aplicación) que se describen a continuación.

14) **panel-registros.component.ts:** Es un componente usado por el panel de administrador. Se encarga de la gestión de los nuevos registros. En su HTML asociado, se carga una lista de las nuevas peticiones de registro, con la opción de aprobar y denegar por cada uno de ellos.

```
aprobarRegistro(usuario : User){

    this._authenticationService.borrarRegistro(usuario).subscribe(
        result=> {

            if(result.resultado == 'ko'){
                this.msg=result.message;
                setTimeout(() => this.visibleAnimate = true);
                this.modalRegistro= true;
            }
            else if(result.resultado == 'ok'){
                this._authenticationService.guardarUsuario(usuario).subscribe(

                    result=>{
                        this.msg=result.message;
                        setTimeout(() => this.visibleAnimate = true);
                        this.modalRegistro= true;
                        for(let i=0; i < this.users.length; i++){
                            if(this.users[i]._id == usuario._id){
                                this.users.splice(i,1);
                                continue;
                            }
                        }
                        for(let i=0; i < this.pagedItems.length; i++){
                            if(this.pagedItems[i]._id == usuario._id){
                                this.pagedItems.splice(i,1);
                                continue;
                            }
                        }
                        if(this.pagedItems.length > 0)
                            this.setPage(this.pager.currentPage);
                        else
                            this.setPage(this.pager.currentPage-1);
                    },
```

Figura 58. Componente panel-registros.component.ts.

15) **panel-gestion-usuarios.ts:** Es otro de los componentes usados por el panel de administración. Su funcionalidad es la de mostrar una vista con los usuarios de sistema y un buscador que ofrece varios filtros para poder listar a través de él. Cada alumno mostrado contiene opciones de administración (modificar datos, cambiar contraseña y eliminar). Su buscador permite filtrar por usuario, nombre y apellidos. A continuación, se muestra el código que permite modificar los datos que el usuario ha cambiado a través del formulario.

```
modificarUsuario(){
     this._authenticationService.updateUser(this.userUpdate).subscribe(

        result=>{
            if(result.respuesta == 'ok'){
                this.cerrarModalModificar();
                this.message="Usuario actualizado correctamente";
                this.modalMessage=true;
                setTimeout(() => this.visibleAnimate = true, 300);

            }else{
                this.cerrarModalModificar();
                this.message="Se ha producido un error actualizando los datos del usuario";
                this.modalMessage=true;
                setTimeout(() => this.visibleAnimate = true, 300);
            }

        },
        error=>{
            this.errorMessage= <any>error;
            if(this.errorMessage != null){
                console.log(this.errorMessage);
                alert('Error en la peticio al servidor para modificar los datos de usario');
            }
        }
    );

}
```

Figura 59. Componente panel-gestion-usuarios.ts.

16) **panel-alumno.component.ts:** Es el componente principal de los alumnos. Se muestra una vista con todas las opciones a las que puede acceder un alumno a través de su pantalla principal. En él, se cargan todas las listas de actividades, clasificadas, al igual que las soluciones aportadas y actividades sin terminar listas para continuar. Además, ofrece dos buscadores, uno para actividades y otro para soluciones, que permite filtrar por numerosos campos. Se muestra a continuación uno de los métodos encargados de cargar un listado de actividades. En este caso las que son propuestas por el profesor con una fecha de fin. El resto de métodos de carga actúan de la misma manera.

```
this._actividadService.getPropuestas().subscribe(
    result =>{
        console.log(result);
        this.propuestas= result.actividades;

        if(!this.propuestas){
            alert('Error en el servidor');
        }
    },
    error => {
        this.errorMessage= <any>error;
        if(this.errorMessage != null){
            console.log(this.errorMessage);
            alert(this.errorMessage);
        }
    }
);
```

Figura 60. Componente panel-alumno.component.ts.

17) **panel-profesor-component.ts:** Al igual que los alumnos, los profesores también disponen de un componente para su pantalla principal, desde donde se gestiona toda la información y opciones disponibles. Aquí se cargan todos los ejercicios y actividades, por categorías, accesibles desde un menú de árbol desplegable. Además, se maneja toda la lógica de creación de actividades y se utiliza el componente de creación de ejercicio descrito anteriormente. Están disponibles también dos buscadores, para ejercicios y actividades, con diferentes filtros de búsqueda. Al inicio del componente se hace una carga de todos los datos que se muestran en el menú de árbol para poder acceder a las listas por categorías. En la pantalla, el profesor puede ir seleccionando los ejercicios que quieren que formen parte de la actividad futura a crear. En la imagen se muestra el método que añade un ejercicio a dicha lista.

```
addActividad(event, id: String){

    let indiceEj= _.findIndex(this.ejersAMostrar, {_id: id});
    if(!this.ejersAMostrar[indiceEj].marcado){
        this.actividad[this.actividad.length]=this.ejersAMostrar[indiceEj];
        this.ejersAMostrar[indiceEj].marcado=true;
    }else{
        let indiceAct= _.findIndex(this.actividad, {_id: id});
        $('.listado-actividad li:eq('+indiceAct+')').removeClass("fadeInLeft").addClass("fadeOut");
        this.sleep(500).then(()=>{
            this.actividad.splice(indiceAct, 1);
            this.ejersAMostrar[indiceEj].marcado=false;
        });
    }
}
```

Figura 61. Componente panel-profesor-component.ts.

18) **resolver-actividad.component.ts:** Uno de los componentes más importante de la aplicación. Se encarga de ofrecer toda la funcionalidad que tiene la pantalla de resolución de una actividad. En él se encuentra la navegación entre los ejercicios, los algoritmos de corrección de la respuesta dada por el alumno, las funcionalidades de encaje de piezas, abrir el diccionario para comprobar palabras, etc… A continuación, se muestra el código empleado para calificar una respuesta a un ejercicio, el cual asigna la nota y un mensaje predefinido.

```
calificar(){
    this.borrarMsgFichas();
    if(this.respuesta == this.actividad[this.ejerSel].solucionPEspanol){
        this.solucion.ejercicios[this.ejerSel].msgCalificacion= this.MS.RESOLVER_RESPUESTA_CORRECTA;
        this.solucion.ejercicios[this.ejerSel].calificacion= 1;
    }else{
        let patron: String[];
        let res: String[];
        res= this.respuesta.split(" ");
        patron= this.actividad[this.ejerSel].solucionFPatron.split(" + ");

        res= _.intersection(res,patron);

       if(_.isEqual(patron, res)){
           this.solucion.ejercicios[this.ejerSel].msgCalificacion=this.MS.RESOLVER_RESPUESTA_NOTA_1;
           this.solucion.ejercicios[this.ejerSel].calificacion= 1;
       }
       else{


           if(res.length == patron.length){
                this.solucion.ejercicios[this.ejerSel].msgCalificacion= this.MS.RESOLVER_RESPUESTA_NOTA_1_2;
                this.solucion.ejercicios[this.ejerSel].calificacion= 1/2;
           }
           else if(res.length > patron.length/2){
                this.solucion.ejercicios[this.ejerSel].msgCalificacion=this.MS.RESOLVER_RESPUESTA_NOTA_1_4;
                this.solucion.ejercicios[this.ejerSel].calificacion= 1/4;
           }else{
                this.solucion.ejercicios[this.ejerSel].msgCalificacion=this.MS.RESOLVER_RESPUESTA_NOTA_0;
                this.solucion.ejercicios[this.ejerSel].calificacion= 0;
           }
       }
    }
    this.solucion.ejercicios[this.ejerSel].respuesta= this.respuesta;
    this.solucion.ejercicios[this.ejerSel].notaProfesor= -1;
    this.solucion.ejercicios[this.ejerSel].msgProfesor="";
    this.resueltos++;
    this.progreso= (this.resueltos * 100) / this.actividad.length;

    if(this.progreso == 100){
        for(var i=0; i < this.solucion.ejercicios.length; i++){
            this.calificacionFinal+=this.solucion.ejercicios[i].calificacion;
        }
        this.solucion.notaFinal=this.calificacionFinal;
        this.terminado=true;
    }
    this.guardarSolucion();
}
```

Figura 62. Componente resolver-actividad.component.ts.

Hasta aquí se ha descrito la estructura principal de la parte cliente de la aplicación. En la siguiente imagen se muestra un diagrama de clases.

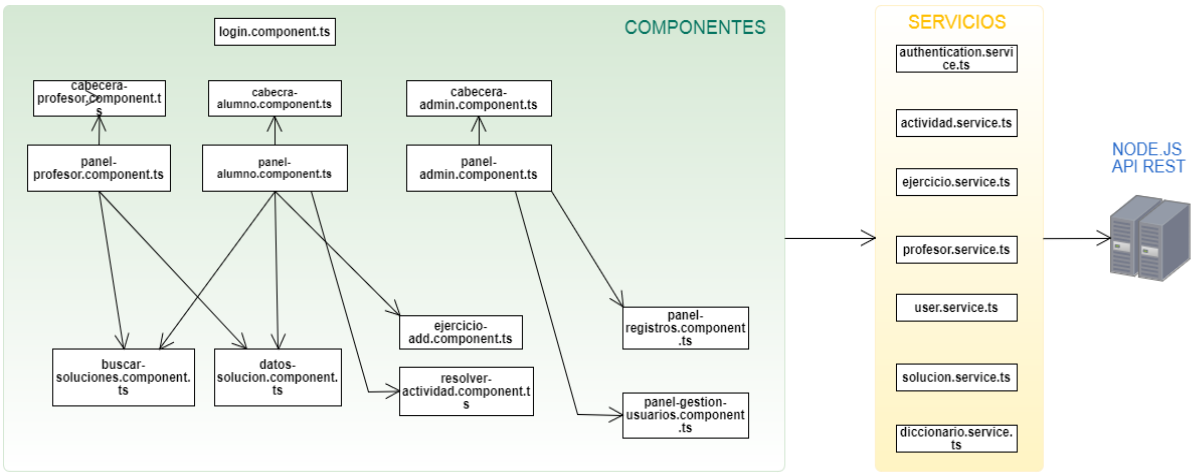


Figura 63. Diagrama clases cliente.

9.2. Implementación del servidor

En el servidor, bajo tecnología NodeJS y Express, se define un sistema de rutas que será accesible desde cualquier cliente. Dichas rutas irán a su vez conectadas a varios controladores, uno por cada entidad del modelo de datos, donde se encuentran todos los métodos encargados del acceso a MongoDB. A continuación, se van a describir las clases que implementan las rutas del API y después cada uno de los controladores con imágenes de algún script de acceso a MongoDB.

1) **routes/actividad.js:** Se definen todas las rutas para interactuar con la entidad de actividad de la BBDD.
2) **routes/auth.js:** Rutas para albergar los métodos y funcionalidad referente a la autenticación y seguridad.
3) **routes/diccionario.js:** Rutas para cargar y obtener el diccionario desde un fichero externo.
4) **routes/ejercicio.js:** Rutas para gestionar la entidad de ejercicio.
5) **routes/profesor.js:** Aunque en sus métodos se accede a la entidad user, se ha separado para implementar métodos referentes a profesores.
6) **routes/solucion.js:** Rutas para gestionar la entidad solución.

Se muestra a continuación una imagen que representa el sistema de rutas de Actividad. El resto de archivos siguen la misma estructura.

```
'use strict'

//Cargamos express
var express= require('express');
//cargamos el controlador
var ActividadController = require('../controllers/actividad');

//Cargamos el router de express
var api= express.Router();


api.get('/actividad/:id', ActividadController.getActividad);
api.get('/cargarActividad/:id', ActividadController.cargarActividad);
api.get('/actividades', ActividadController.getActividades);
api.post('/actividad', ActividadController.saveActividad);
api.get('/actividad-disponibles', ActividadController.getDisponibles);
api.get('/actividad-disponiblesNB', ActividadController.getDisponiblesNInicial);
api.get('/actividad-disponiblesNM', ActividadController.getDisponiblesNMedio);
api.get('/actividad-disponiblesNA', ActividadController.getDisponiblesNAlto);
api.get('/actividad-propuestas', ActividadController.getPropuestas);
api.get('/actividad-propuestasByApertura', ActividadController.getPropuestasByApertura);
api.get('/actividad-propuestasByCierre', ActividadController.getPropuestasByCierre);
api.get('/actividad-idProfesorDisp/:id', ActividadController.getByIdProfesorDisp);
api.get('/actividad-idProfesorProp/:id', ActividadController.getByIdProfesorProp);
api.get('/actividad-miColeccion/:id_profesor', ActividadController.getActsMiColeccion);
api.get('/actividad-miColeccionNivelA/:id_profesor', ActividadController.getActsMiColeccionNivelA);
api.get('/actividad-miColeccionNivelM/:id_profesor', ActividadController.getActsMiColeccionNivelM);
api.get('/actividad-miColeccionNivelB/:id_profesor', ActividadController.getActsMiColeccionNivelB);
api.get('/actividad-actVisibles/:id_profesor', ActividadController.getActsVisibles);
api.get('/actividad-actVisiblesNivelA/:id_profesor', ActividadController.getActsVisiblesNivelA);
api.get('/actividad-actVisiblesNivelM/:id_profesor', ActividadController.getActsVisiblesNivelM);
api.get('/actividad-actVisiblesNivelB/:id_profesor', ActividadController.getActsVisiblesNivelB);
api.get('/actividad-actNoVisibles/:id_profesor', ActividadController.getActsNoVisibles);
api.get('/actividad-actNoVisiblesNivelA/:id_profesor', ActividadController.getActsNoVisiblesNivelA);
api.get('/actividad-actNoVisiblesNivelM/:id_profesor', ActividadController.getActsNoVisiblesNivelM);
api.get('/actividad-actNoVisiblesNivelB/:id_profesor', ActividadController.getActsNoVisiblesNivelB);
api.get('/actividad-otrasColecciones/:id_profesor', ActividadController.getActsOtrasColecciones);
api.get('/actividad-otrasColeccionesNivelA/:id_profesor', ActividadController.getActsOtrasColeccionesNivelA);
api.get('/actividad-otrasColeccionesNivelM/:id_profesor', ActividadController.getActsOtrasColeccionesNivelM);
api.get('/actividad-otrasColeccionesNivelB/:id_profesor', ActividadController.getActsOtrasColeccionesNivelB);
api.put('/actividad/:id', ActividadController.updateActividad);
api.put('/actividad-ejercicio/:id', ActividadController.borrarEjercicio);
api.delete('/actividad/:id', ActividadController.deleteActividad);
api.post('/actividades-byCriteria', ActividadController.getActividadesByCriteria);

module.exports= api;
```

Figura 64. Estructura de rutas para solución.js.

Como se puede comprobar en cada una de las rutas, estas se comunican con cada uno de los controladores. Dichos controladores se encuentran bajo el directorio *controllers*, y tienen asignado el mismo nombre que aparece bajo el directorio de rutas.

Los controladores, como ya hemos comentado, se encargan de mapear la relación con la entidad correspondiente en MongoDB, y alberga todos los métodos de acceso a esta. Se aprecia a continuación uno de los métodos contenidos en el controlador de actividad, encargado de cargar una actividad a través de su id.

```
function cargarActividad(req, res){

    var actividadId= req.params.id;
    Actividad.findById(actividadId, function(err, actividad){

        if(err){
            res.status(500).send({message:'Error al devolver el ejercicio'});
        }
        else{
            if(!actividad){
                res.status(404).send({message:'No existte la actividad'});
            }
            else{
                Ejercicio.populate(actividad, {path:"ejercicios"}, function(err,actividad){
                    res.status(200).send({actividad});
                });
            }
        }
    });
}
```

Figura 65. Controlador de actividad.

Cada uno de los controladores definidos, hace uso de las entidades definidas en MongoDB. A continuación, para seguir el ejemplo, se muestra una imagen del modelo actividad que se mapea con la colección en MongoDB gracias a la configuración del Framework Mongoose.

```
'use strict'

var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var ActividadSchema = Schema({

    titulo: String,
    id_profesor: String,
    profesor: String,
    fecha_creacion: Date,
    nivel: String,
    ejercicios: [{type:Schema.ObjectId, ref: "Ejercicio"}],
    visible: Boolean,
    propuesta: Boolean,
    fecha_prop_fin: Date
});

module.exports= mongoose.model('Actividad', ActividadSchema, 'actividades');
```

Figura 66. Modelo actividad.

Para que todo esto sea accesible como un API REST, hay un módulo Javascript encargado de configurar el sistema de rutas gracias al Framework Express. Dicho módulo se encuentra en el directorio principal de la aplicación servidor, llamado app.js.

```
'use strict'
var api = require('./routes/ejercicio');
var api2= require('./routes/actividad');
var apiProfesor= require('./routes/profesor');
var apiDiccionario= require('./routes/diccionario');
var apiSolucion= require('./routes/solucion');
var apiAuth= require('./routes/auth');

var bodyParser= require('body-parser');
var express= require('express');
var app= express();
var cors = require('cors');
var authCtrl = require('./auth');
var middleware = require('./middleware');

app.use(bodyParser.urlencoded({extended:false}));
app.use(bodyParser.json());

app.use(cors());

app.use((req, res, next) =>{

    res.header('Access-Control-Allow-Origin', '*');
    //res.header('Access-Control-Allow-Credentials', false);
    res.header('Access-Control-Allow-Headers', 'X-API-KEY, Origin, X-Requested-With, Content-Type, Accept, Access-Control-Request-Method');
    res.header('ccess-Control-Allow-Methods', 'GET, POST, OPTION, PUT, DELETE');
    res.header('Allow', 'GET, POST, OPTION, PUT, DELETE');
    next();
});

app.use('/api', api);
app.use('/api2', api2);
app.use('/apiProfesor', apiProfesor);
app.use('/apiDiccionario', apiDiccionario);
app.use('/apiSolucion', apiSolucion);
app.use('/apiAuth', apiAuth);

module.exports= app;
```

Figura 67.Módulo app.js.

Como se ve en la imagen, se importan las dependencias del Framework, se definen las cabeceras http y los nombres de cada una de las diferentes estructuras de rutas.

10. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se ha desarrollado una aplicación e-learning que implementa una metodología de aprendizaje de latín. La aplicación facilita al docente la creación de ejercicios y actividades siguiendo la metodología mencionada. Así mismo, la aplicación permite el registro de estudiantes para poder realizar los ejercicios propuestos y facilita retroalimentación de la evaluación de dichos los mismos.

En la aplicación desarrollada cabe destacar algunas características tales como el ofrecer una interface amigable y usable por cualquier persona, la facilidad de navegación sobre las diferentes páginas, y la manera intuitiva y visual de realizar las actividades propuestas mediante piezas de rompecabezas.

El diseño de la aplicación garantiza la posibilidad de extender de una manera fácil la funcionalidad en un futuro.

Las principales dificultades surgidas durante la realización del trabajo han sido la falta de un conocimiento profundo de las tecnologías utilizadas y el trabajo en grupo dado que los horarios de trabajo y las responsabilidades particulares de cada integrante hacía complicado llevar a cabo reuniones y coordinarse adecuadamente. En cualquier caso, el sentimiento general del grupo ha sido de satisfacción por el trabajo realizado.

Las principales líneas de trabajo futuro son:

- El desarrollo de una aplicación móvil para que los usuarios puedan hacer su evaluación desde sus smartphones o tablets.

- Aumentar el número de tipo de ejercicios que se pueden crear para dar un mayor dinamismo a las actividades
- Permitir el cambio de idioma de las interfaces para una mayor usabilidad por parte de personas de todo el mundo.
- Adaptar la interfaz para el uso de personas con problemas de visión reducida.

10. CONCLUSSIONS AND FUTURE WORK

In this work an e-learning application has been developed that implements a Latin learning methodology. The application facilitates the teacher to create exercises and activities following the methodology mentioned. Also, the application allows the registration of students to be able to perform the proposed exercises and facilitates the feedback of the evaluation of said exercises.

In the developed application it is possible to emphasize some characteristics such as to offer a friendly interface and useable by any person, the ease of navigation on the different pages, and the intuitive and visual way to carry out the proposed activities using pieces of puzzles.

The design of the application guarantees the possibility of easily extending the functionality in the future.

The main difficulties encountered during the work were the lack of an in-depth knowledge of the technologies used and the group work since the work schedules and the particular responsibilities of each member made it difficult to hold meetings and coordinate properly. In any case, the general feeling of the group has been of satisfaction by the work done.

The main lines of future work are:

- The development of a mobile application so that users can make their evaluation from their smartphones or tablets.
- Increase the number of types of exercises that can be created to give more dynamism to activities
- Allow the language change of the interfaces for greater usability by people around the world.
- Adapt the interface for the use of people with reduced vision problems.

11. CONTRIBUCIONES

Alberto Daimiel Blanco: Desarrollo de estructura del proyecto, enrutadores y servicios que se comunican con el servidor. Contribuciones en el desarrollo del panel de profesor. Desarrollo del componente de resolver actividad en alumno. Desarrollo de sistema de login y aportaciones en panel de Administrador.

Mounir Hichou: Desarrollo del plugin para extraer a nuestro sistema el diccionario creado y aportado por la facultad de Filología. Contribuciones en el desarrollo del API REST en NodeJS y desarrollo de la capa de acceso a datos.

Darío Simón: Desarrollo del panel de Alumno, y definición de rutas y servicios para conectarse con el servidor NodeJS. Contribuciones en el desarrollo del API REST.

Luis Zorrilla: Contribuciones al panel de profesor y al panel del Administrador del sistema. Definición de servicios para conexión con el servidor NodeJS.

REFERENCIAS

[1] Angular2: http://www.angular2.com/

[2] Bootstrap: http://getbootstrap.com/

[3] CSS: https://www.desarrolloweb.com/manuales/css3.html

[4] ExeLearning: http://exelearning.net/

[5] Hot Potatoes: https://hotpot.uvic.ca/

[6] HTML5: https://developer.mozilla.org/es/docs/HTML/HTML5

[7] JClic: http://clic.xtec.cat/es/jclic/

[8] JQuery: https://api.jquery.com

[9] LearningApps.org: https://learningapps.org/

[10] Librería drag&drog: https://www.html5rocks.com/es/tutorials/dnd/basics/

[11] LingQ: https://www.lingq.com/

[12] Linguim: http://linguim.com/es/la/

[13] MongoDB: https://www.mongodb.com/es

[14] Node.js: https://nodejs.org/es/

[15] Typescript: https://www.typescriptlang.org/

[16] Underscore: http://underscorejs.org

ANEXO I: MANUAL DE INSTALACIÓN DE LA APLICACIÓN.

Instalar Cygwin

Accederemos al sitio oficial de Cygwin https://www.cygwin.com/ y seleccionaremos la opción de acuerdo con nuestro sistema (32 o 64 bits) en el apartado current cygwin dll versión de la página.

Una vez descargado el archivo de instalación ejecutaremos el instalador y siguiendo las opciones que se adecuen a nuestra necesidad.

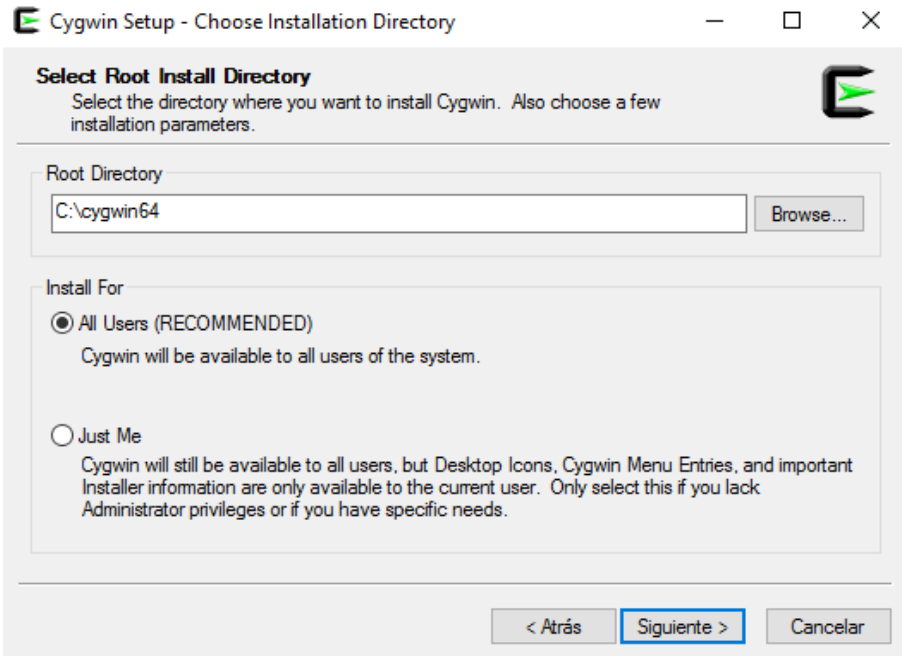


Figura 68. Instalación cygwin 1.

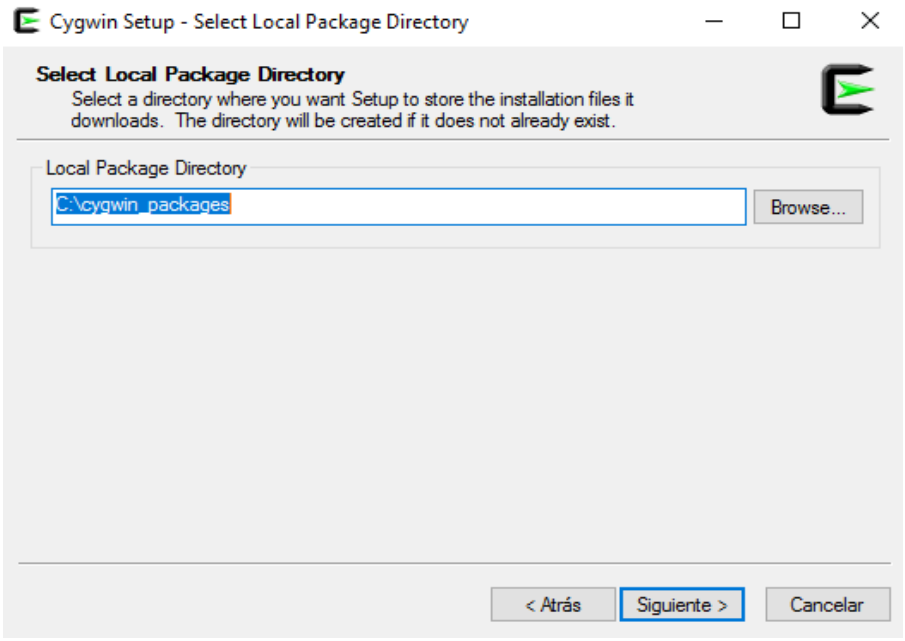


Figura 69. Instalación cygwin 2.

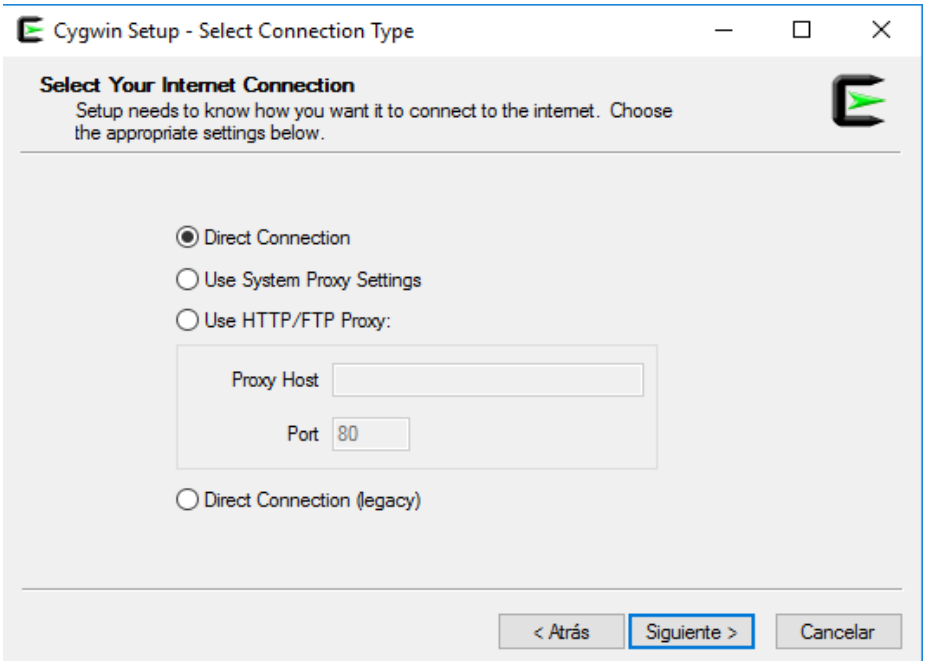


Figura 70. Instalación de cygwin 3.

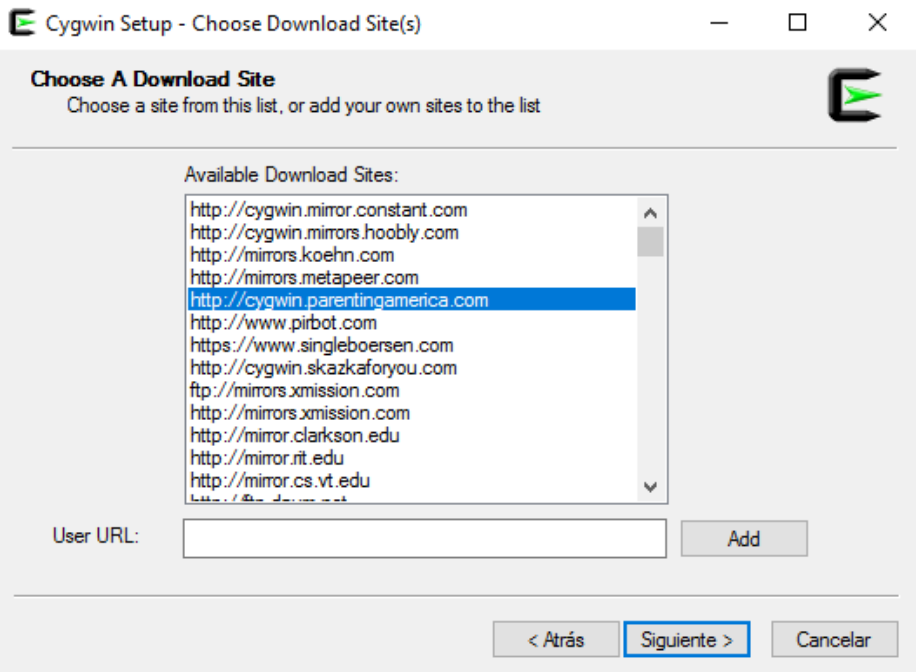


Figura 71. Instalación de cygwin 4.

Instalar Node JS

Accederemos a la página de nodejs https://nodejs.org/es/ para descargar el archivo de instalación.

Figura 72. Instalación de Node 1.

Una vez descargado seguiremos los pasos indicados por el instalador.

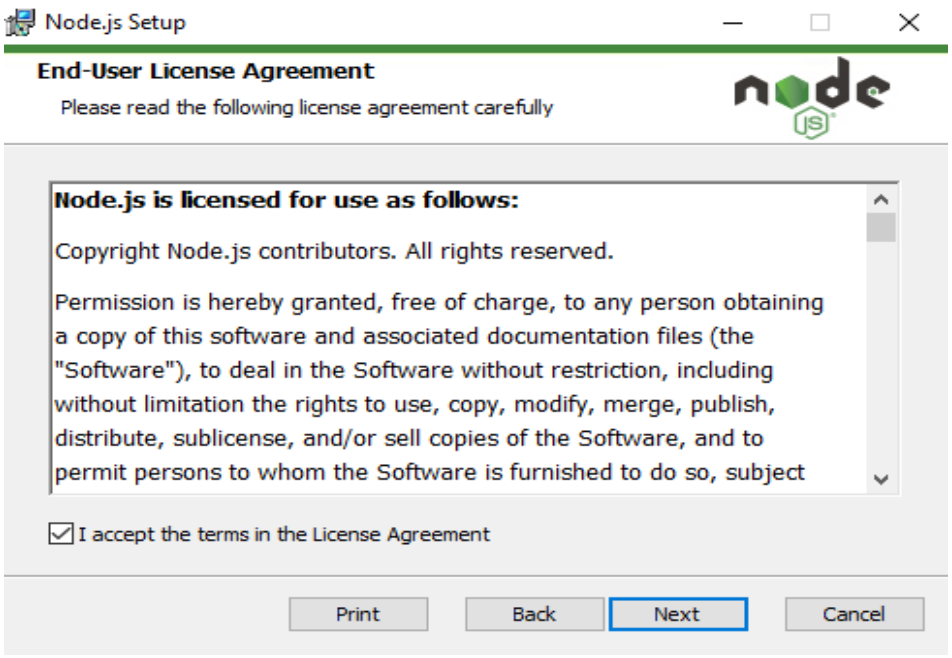


Figura 73. Instalación de Node 2.

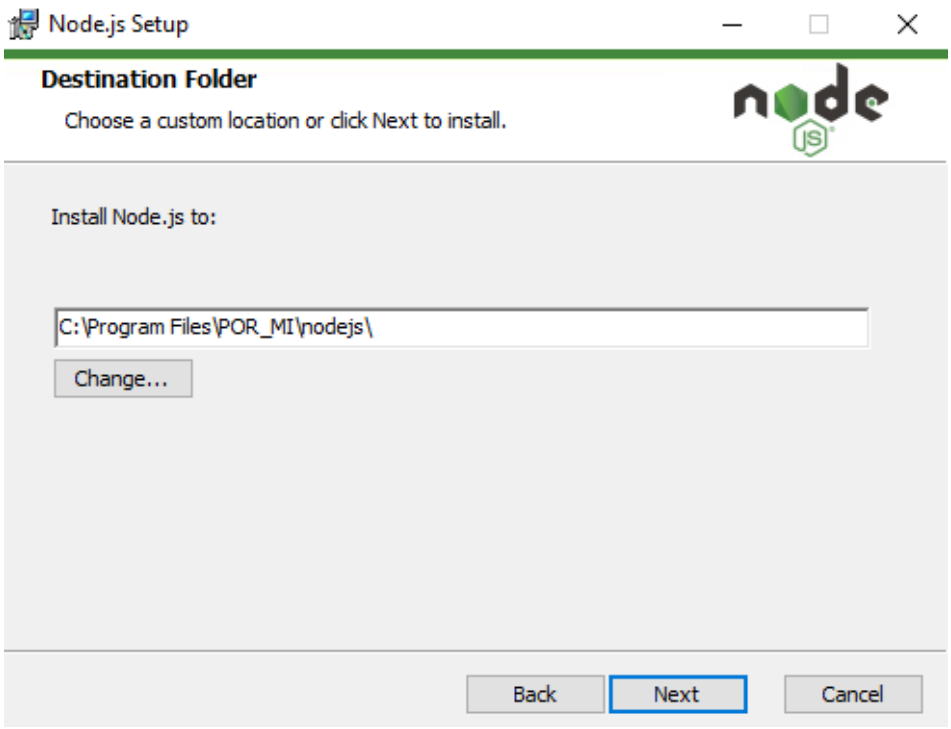


Figura 74. Instalación de Node 3.

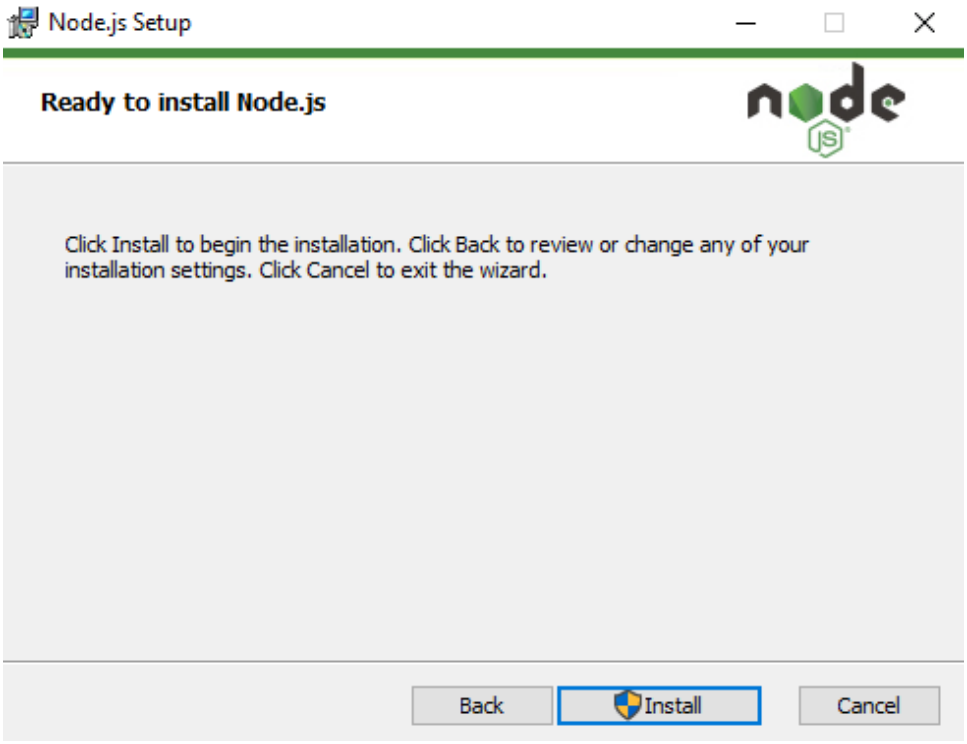


Figura 75. Instalación de Node 4.

Instalar MongoDB

Al igual que con los anteriores, entrando a la página oficial de mongodb descargamos el archivo de instalación.

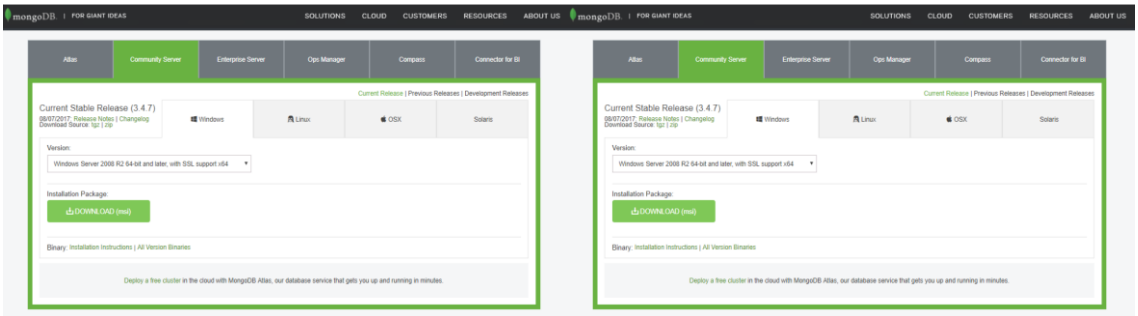


Figura 76. Instalación de MongoDB 1.

Una vez descargado seguiremos las indicaciones del instalador.

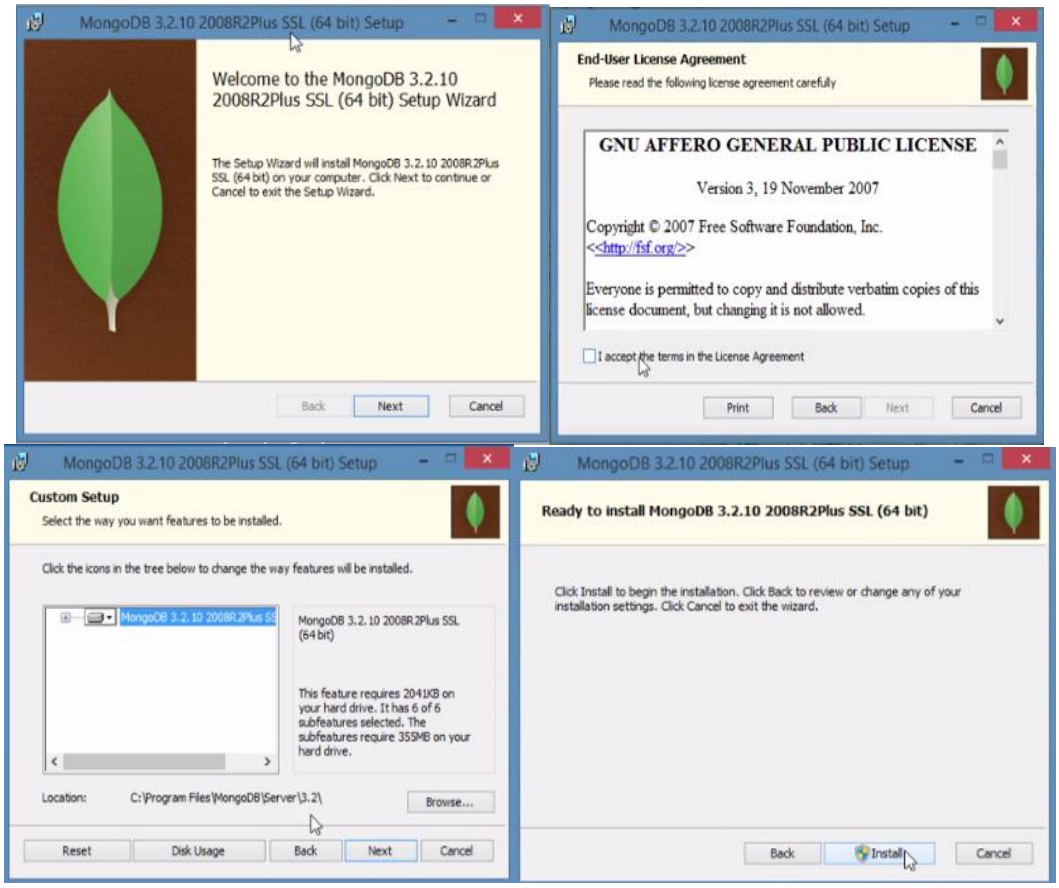


Figura 77. Instalación de MongoDB 2.

Después de descargar e instalar el programa debemos crear una carpeta llamada "data" en el directorio C: con una carpeta en su interior que se debe llamar "db".

Ejecutar mongod.exe y mongo.exe del directorio de mongo para la utilización de la bbdd.

ANEXO II: MANUAL DE USUARIO

Login

Esta es la pantalla que se muestra al abrir la aplicación desde aquí se puede acceder a la misma introduciendo su usuario y contraseña o se puede acceder a la página de registro.

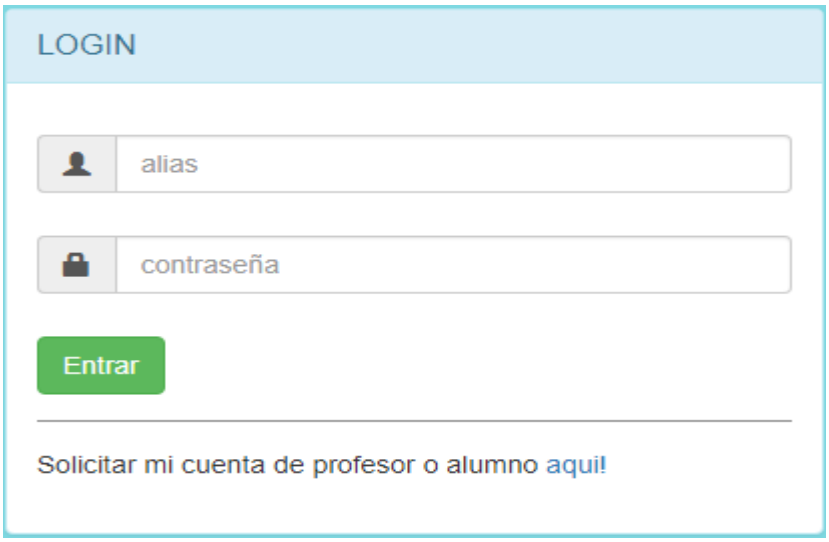


Figura 78. Login.

Registro

En esta pantalla los usuarios podrán introducir sus datos y enviar una solicitud de registro que quedará pendiente de que la acepte o rechace el administrador.

Figura 79. Registro.

PROFESOR

Panel de profesor

Al acceder un profesor se mostrará esta pantalla donde podrá realizar las acciones deseadas correspondientes a su rol y se mostrará el listado de ejercicios o de actividades según seleccione en el árbol de búsqueda.

Figura 80. Panel de profesor.

Crear actividad

Seleccionando los ejercicios desde el listado (pulsando el botón de “Añadir a actividad”), estos pasarán a ser parte de la actividad. Los ejercicios que quedan apilados para formar parte de la futura actividad, pueden ser reordenados pinchando y arrastrando a la posición deseada. Una vez seleccionados los ejercicios, al pulsar el botón crear actividad, se deberá introducir el nombre el tipo y la visibilidad de la misma.

Figura 81. Crear actividad 1.

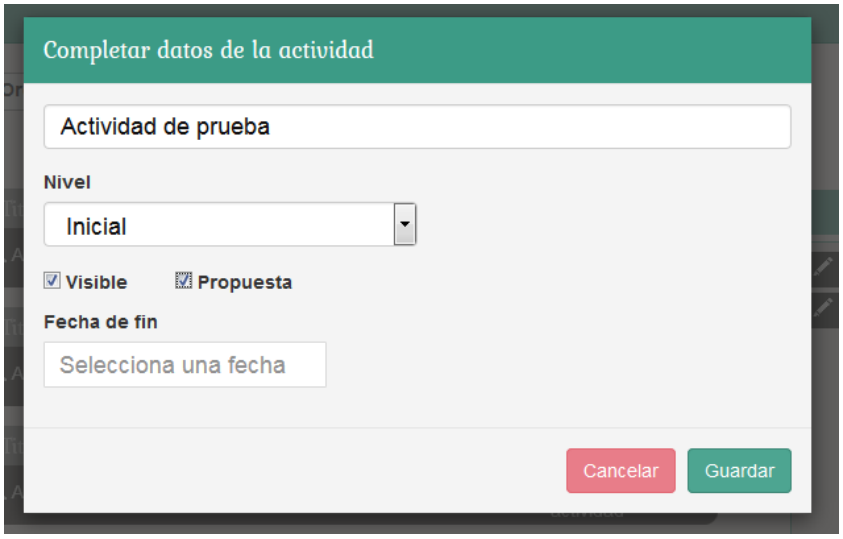


Figura 82. Crear actividad 2.

Árbol de búsqueda

Aquí el profesor podrá decidir qué listado se muestra y filtrarlo por el nivel seleccionado. Se mostrará el total y el número de ejercicios/actividades de cada nivel que hay creados

Figura 83. Árbol de busqueda.

Desplegable acciones

Desplegable desde el que se podrá acceder al creador de ejercicios, así como al buscador de soluciones.

Figura 84. Despliegue de acciones.

Crear ejercicio

Pantalla de creación de ejercicios accesible desde el desplegable de acciones.

Figura 85. Crear ejercicio.

Ver soluciones

Pantalla donde el profesor podrá consultar las soluciones de actividades propuestas enviadas por los alumnos para su corrección, permite el filtrado por actividad, alumno o por fechas propuestas. El listado de soluciones se mostrará a la derecha del recuadro permitiendo al profesor seleccionar la deseada.

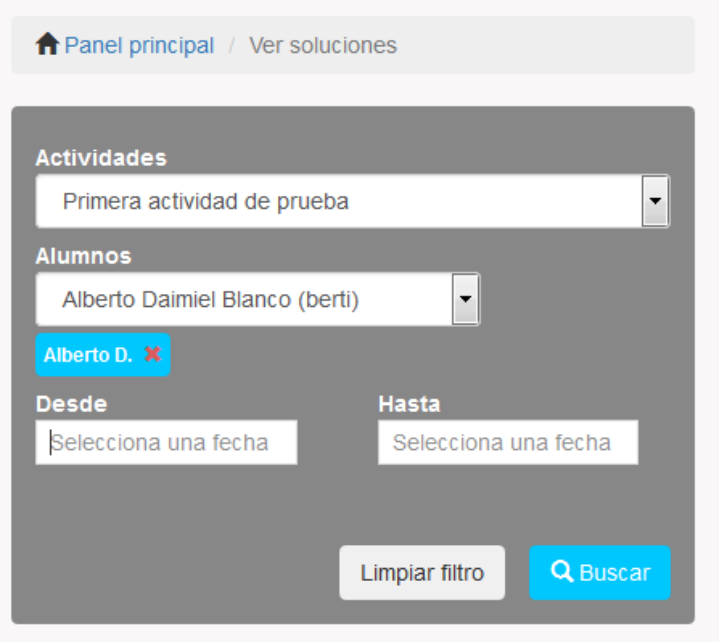


Figura 86. Ver soluciones.

Buscadores

Permiten una búsqueda más filtrada de ejercicios, así como de actividades

Figura 87. Buscador ejercicios.

Figura 88. Buscador actividades.

Modificar ejercicios

En el listado de ejercicios clicando sobre el icono del engranaje se podrá modificar ciertos aspectos del ejercicio creado. Esta opción sólo es accesible si el ejercicio ha sido creado por el profesor logueado.

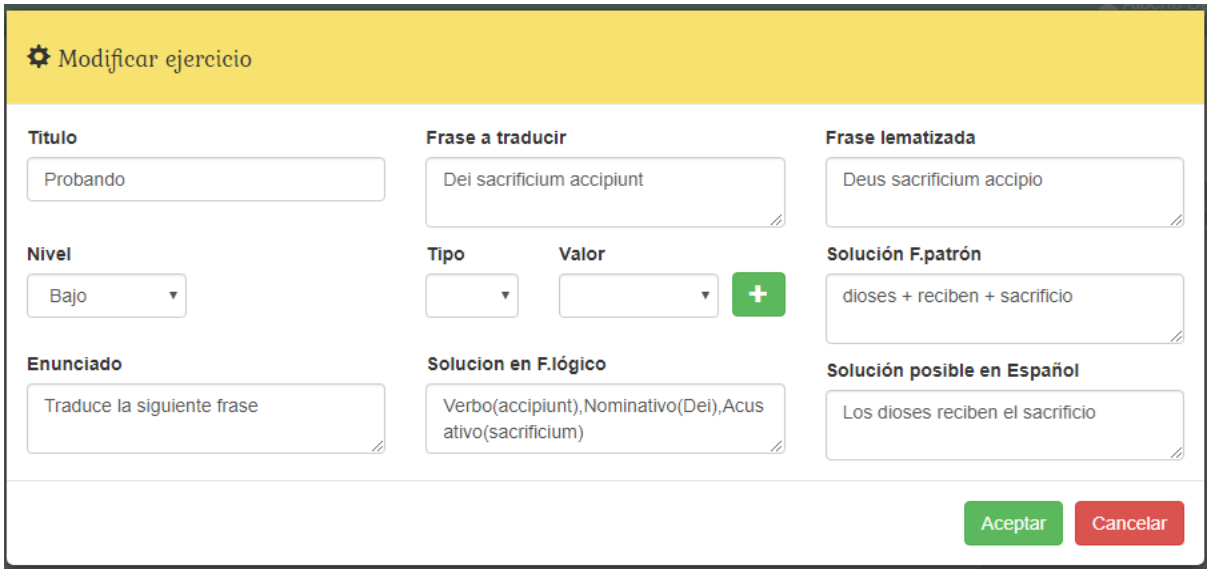


Figura 89.Modificar ejercicio.

Modificar actividad

Al igual que con ejercicios al listar las actividades las que hayan sido creadas por el profesor logueado podrán ser modificadas permitiendo añadir o quitar ejercicios. El panel cambiará de color para hacer más evidente que se encuentra en fase de modificación de una actividad. Se permite navegar por el panel de profesor pudiendo listar nuevamente los ejercicios sin salir de la modificación hasta que no se cliquee sobre cancelar. Esto permite añadir más ejercicios que no formaran parte de la actividad. Además, es posible cambiar los ejercicios de orden arrastrando en la posición deseada.

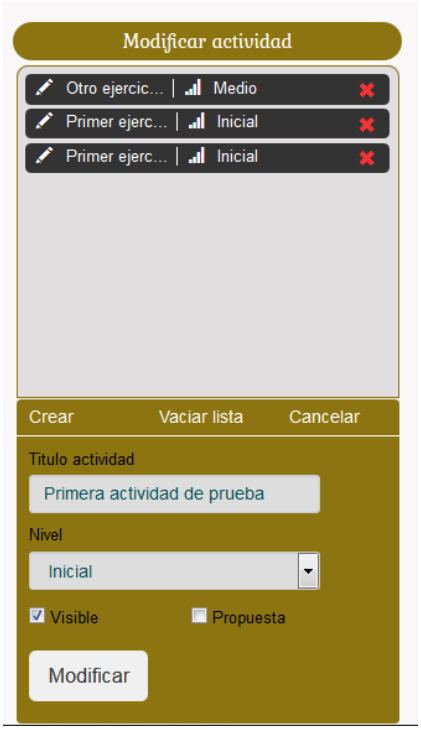


Figura 90. Modificar actividad.

ALUMNO

Panel de alumnos

Al acceder un alumno se mostrará esta pantalla donde podrá realizar las acciones deseadas correspondientes a su rol y se mostrará el listado de sus soluciones y actividades propuestas según seleccione en el árbol de búsqueda.

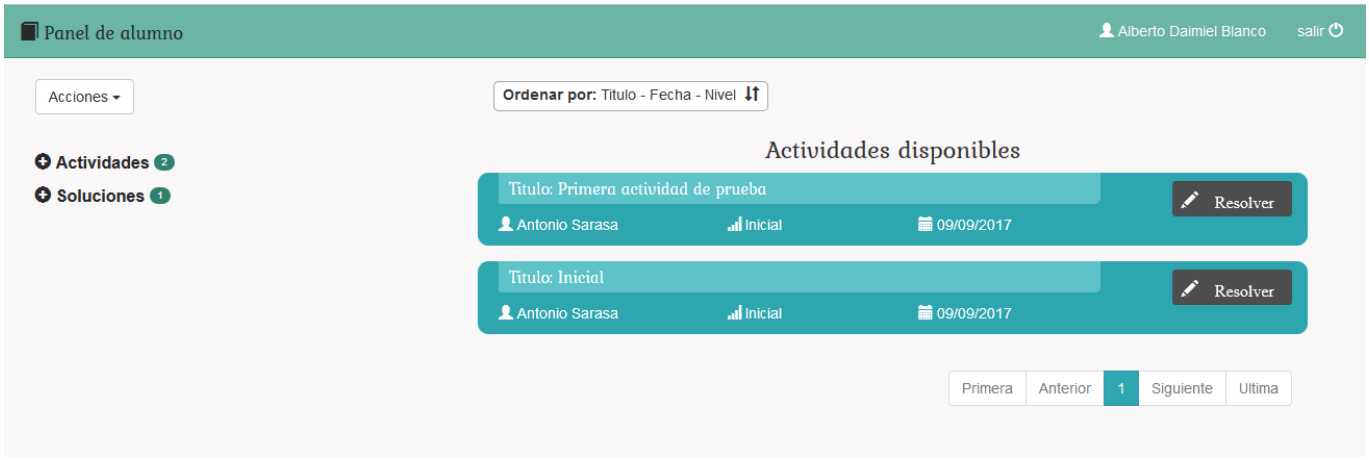


Figura 91. Panel de alumnos.

Árbol de búsqueda

Permitirá al alumno listar tanto las actividades como sus soluciones ya sean terminadas o pendientes de terminar ya que tienen la posibilidad de dejar una actividad a medio hacer para continuar su solución más adelante.

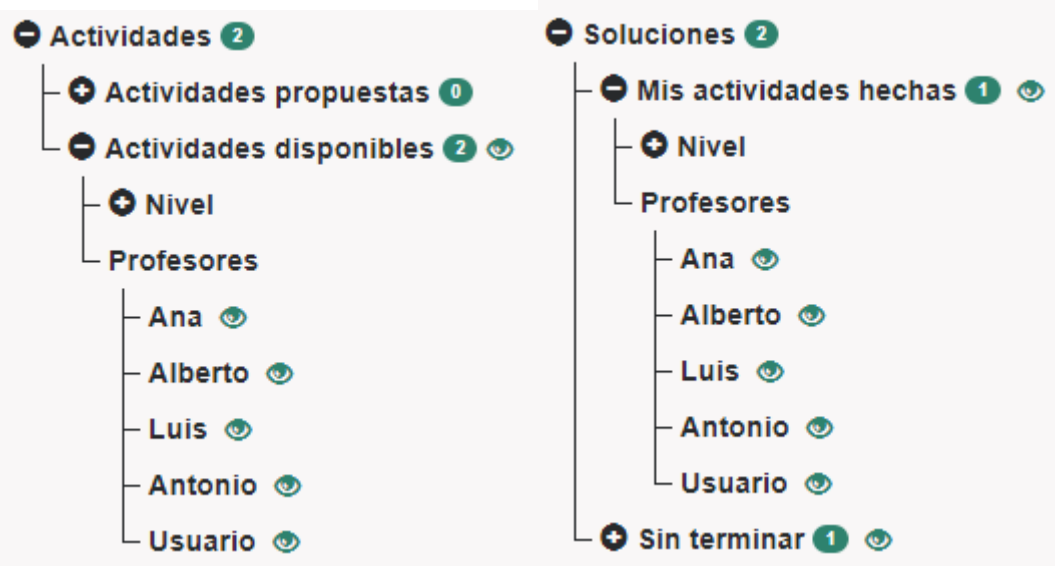


Figura 92. Árboles de búsqueda de alumnos.

Resolver actividad

La pantalla de resolución de actividades permitirá al alumno resolver los ejercicios que componen dicha actividad. Dispondrá de las fichas, las cuales, deberá arrastrar hasta la palabra que crea que les corresponde, quedando esta marcada con el color correspondiente cuando es un acierto. También podrá acceder al diccionario para consultar la información sobre las palabras que aparecen en la frase a resolver. Además, le permitirá navegar por los ejercicios sin necesidad de haberlos terminado, así como guardar la actividad sin haber terminado o enviarla para su corrección por parte del profesor cuando haya completado al 100% los ejercicios propuestos. Durante la realización de la actividad, se muestra una barra de progreso que indica el porcentaje realizado. El alumno puede consultar información sobre el ejercicio. También podrá consultar la respuesta correcta tras calificar cada uno de ellos. Cada vez que el alumno enviar un ejercicio para calificar, la aplicación le muestra un mensaje con la calificación obtenida y un mensaje que acompaña a dicha calificación.

Figura 93. Resolver actividad.

Abrir diccionario

Al abrir el diccionario el alumno podrá consultar las palabras que forman la frase del ejercicio con un resumen detallado de sus características (Significado, lema, caracterización etc...) con lo que poder guiarse para la comprensión y resolución del ejercicio. Podrá hacer click sobre cada una de las palabras que se proponen para el ejercicio en el que se encuentra. Además podrá usar el diccionario para hacer búsquedas sobre cualquier palabra que se aloje en el diccionario de latín.

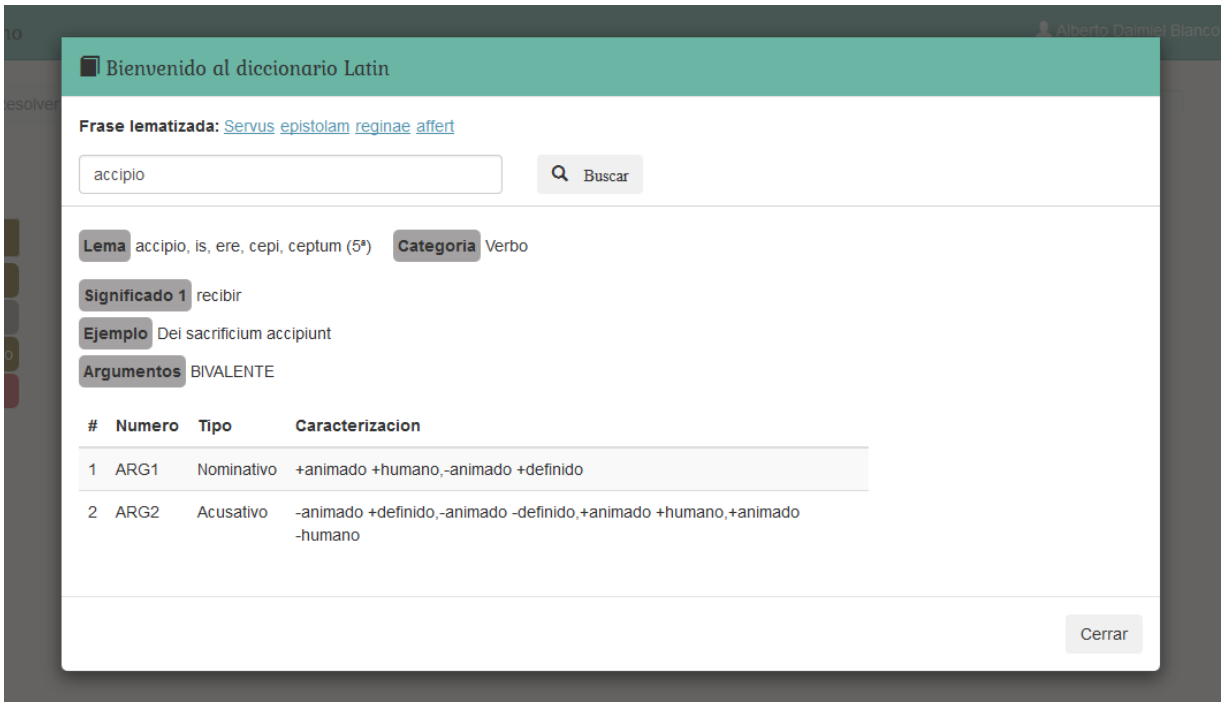


Figura 94. Abrir diccionario.

Ver solución

Al listar las soluciones y clicar sobre la acción ver, se redirigirá a esta pantalla donde el alumno podrá ver su respuesta a cada uno de los ejercicios que componen la actividad. Se mostrará la información y nota global de la actividad. Además, por cada ejercicio se mostrará la respuesta del alumno, la nota y mensaje que la aplicación asignó automáticamente y opcionalmente, un mensaje que podrá dejar el profesor. El profesor también tiene la opción de cambiar la puntuación asignada a cada ejercicio.

Figura 95. Ver solución.

ADMINISTRADOR

Panel principal

Al loguearse un administrador accederá a esta pantalla donde podrá seleccionar cuál de sus acciones disponibles desea realizar. Para las opciones de gestión de registros y usuarios se muestra el número que compone la lista para aportar mas información.

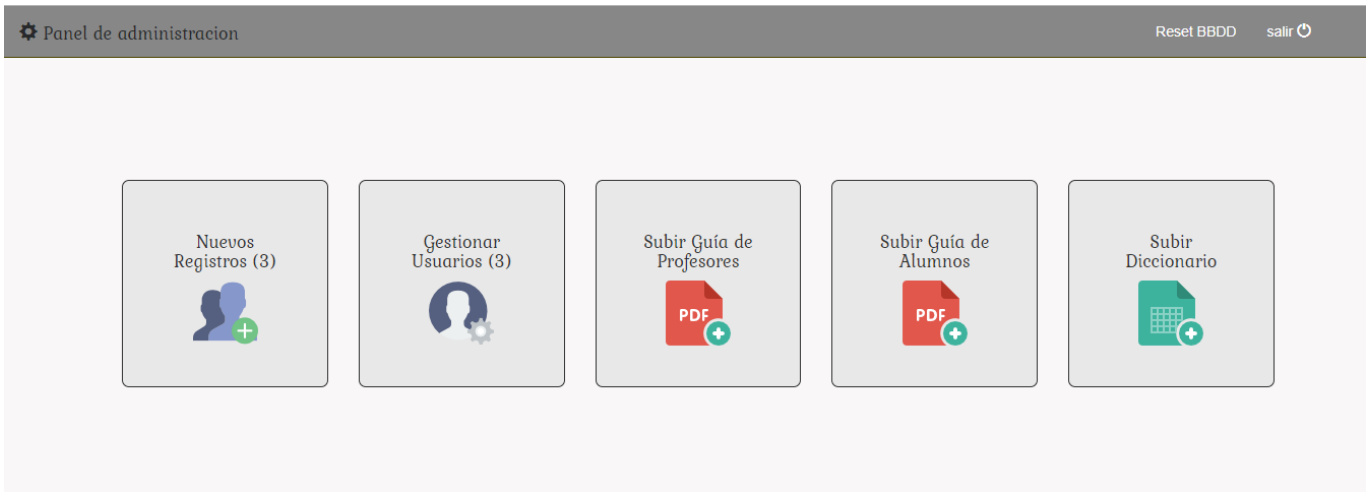


Figura 96. Panel de administrador.

Nuevos registros

En la opción nuevos registros en administrador tendrá encolados las peticiones de registro de los usuarios pudiendo el rechazarlas o aceptarlas. Haciendo click sobre cada una de las fichas aparecerá una ventana modal con todos los datos rellenados por el usuario.

Figura 97. Nuevos registros.

Gestionar usuarios

En la opción de gestión de usuarios el administrador podrá cambiar datos de la persona registrada, su contraseña o eliminarlos según seleccione la opción en el desplegable de opciones. Dispone de un buscador con filtros. Al hacer click sobre “Borrar búsqueda”, se volverán a cargar todos los usuarios del sistema.

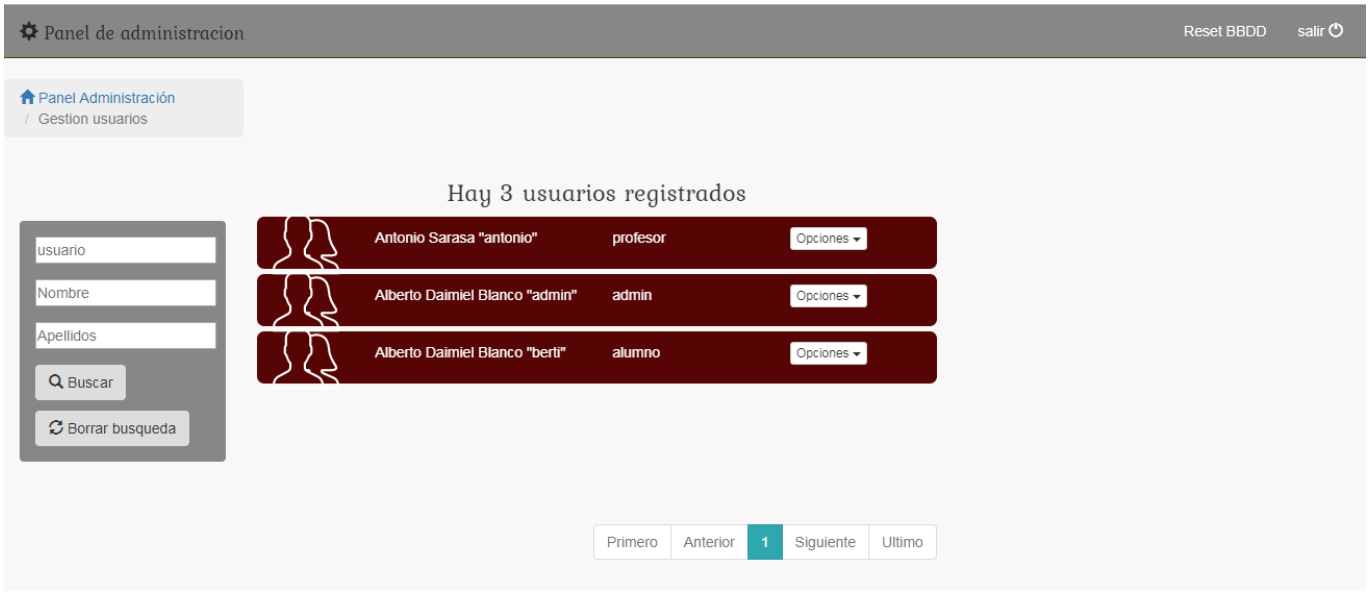


Figura 98. Gestionar usuarios.

Cargar manuales y diccionario

Las tres siguientes opciones cuentan de los mismos paneles y sirven para cargar manuales de usuario en los perfiles de los usuarios (profesores o alumnos) o para cargar un diccionario específico. Al seleccionar alguna de estas opciones se abrirá una ventana modal pidiendo el archivo correspondiente, se facilitará un explorador de archivos para facilitar el trámite.

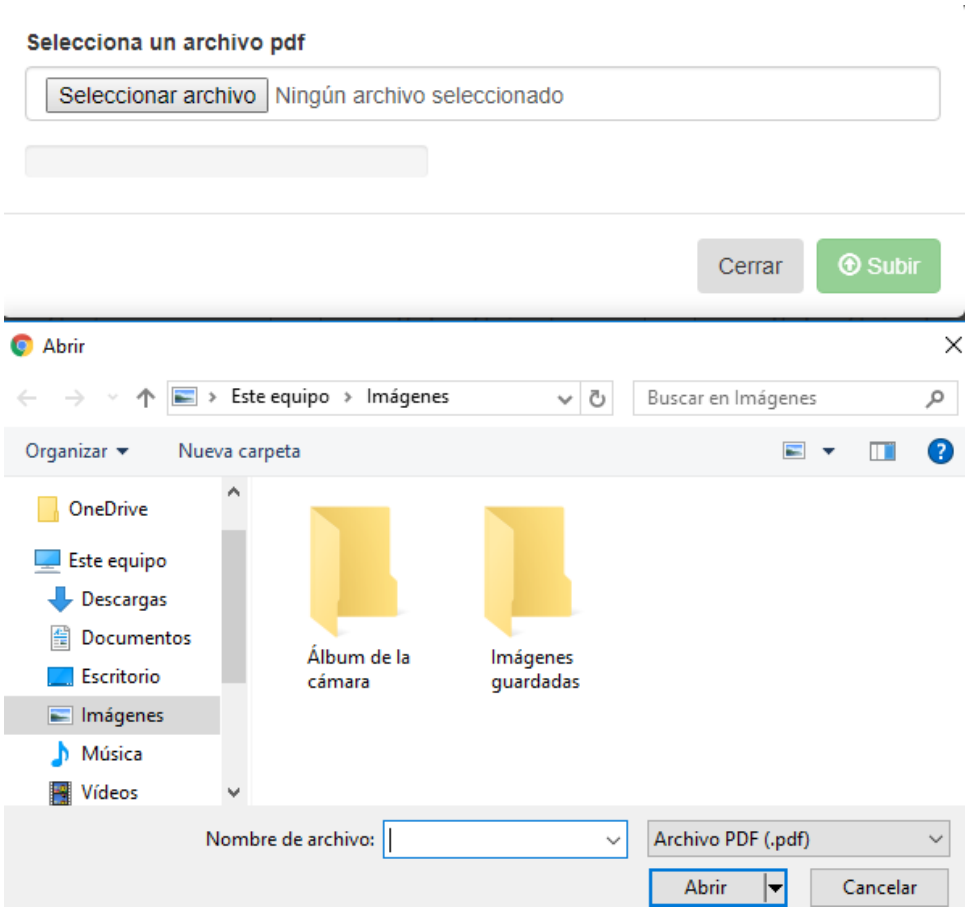


Figura 99. Cargar manuales.